INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT

INPUT

2

Editor
VICTOR CIVITA

REDAÇÃO
Diretora Editorial: Iara Rodrigues

Editor Executivo: Antonio José Filho

Editor Chefe: Paulo de Almeida
Editor de Texto: Cláudio A.V. Cavalcanti
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Ailton Oliveira Lopes,
Dilvacy M. Santos, Grace Alonso Arruda, José Maria de Oliveira,
Monica Lenardon Corradi
Secretária de Redação/Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström,
José Benedito de Oliveira Damião, Maria de Lourdes Carvalho,
Marisa Soares de Andrade, Mauro de Queiroz

COLABORADORES
Consultor Editorial Responsável: Dr. Renato M.E. Sabbatini
(Diretor do Núcleo de Informática Biomédica da Universidade
Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática
Ltda., Campinas, SP
Tradução: Reinaldo Cúrcio
Tradução, adaptação, programação e redação:
Abílio Pedro Neto, Aluísio J. Dornellas de Barros,
Marcelo R. Pires Therezo, Raul Neder Porrelli
Coordenação Geral: Rejane Felizatti Sabbatini
Editora de Texto: Ana Lúcia B. de Lucena
Assistente de Arte: Dagmar Bastos Sampaio

COMERCIAL
Diretor Comercial: Roberto Martins Silveira

Gerente Comercial: Flávio Maculan
Gerente de Circulação: Denise Maria Mozol

PRODUÇÃO
Gerente de Produção: João Stungis

Coordenador de Impressão: Atílio Roberto Bonon
Preparador de Texto/Coordenador: Eliel Silveira Cunha
Preparadores de Texto: Alzira Moreira Braz,
Ana Maria Dilguerian, Karina Ap. V. Grechi, Levon Yacubian,
Luciano Tasca, Maria Tereza Galluzzi, Maria Tereza Martins Lopes,
Paulo Felipe Mendrone
Revisor/Coordenador: José Maria de Assis
Revisoras: Conceição Aparecida Gabriel,
Isabel Leite de Camargo, Ligia Aparecida Ricetto,
Maria de Fátima Cardoso, Nair Lucia de Britto
Paste-up: Anastase Potaris, Balduino F. Leite, Edson Donato

Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, nº 2000 - 3º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da lei 5 988, de 14/12/1973).
Esta obra foi composta na AM Produções Gráficas Ltda.
e impressa na Divisão Gráfica da Editora Abril S.A.

SUMÁRIO

APLICAÇÕES

CÓDIGO DE MÁQUINA

PERIFÉRICOS

PROGRAMAÇÃO BASIC

PROGRAMAÇÃO DE JOGOS

RELAÇÕES MUITO LÓGICAS

- OPERADORES RELACIONAIS
- MAIOR, MENOR, IGUAL
- OPERADORES LÓGICOS
- AND, OR E NOT
- TABELAS-VERDADE

Os computadores são capazes de executar milhões de operações lógicas em apenas um segundo. Mas a lógica é também parte fundamental de qualquer programa. Conheça seus operadores.

A programação BASIC utiliza três tipos de expressão: aritmética, de cadeia e lógica. As duas primeiras compõem a maior parte de qualquer programa. Porém, cabe às expressões lógicas a responsabilidade pelas decisões ao longo de um programa.

A função das expressões lógicas é, simplesmente, verificar se algo é "verdadeiro" ou "falso". As palavras e símbolos usados para isso em um programa são chamados de *operadores* (ou, ainda, *conectores*).

Nas expressões lógicas empregam-se dois tipos de operadores: os *operadores relacionais* (que incluem símbolos matemáticos como <, > e =) e os *operadores lógicos*: **AND**, **OR** e **NOT** (incluídos na lista de comandos de qualquer versão BASIC).

MAIOR, MENOR, IGUAL

Os operadores relacionais podem ser usados até no mais simples dos programas em BASIC. As comparações possíveis são:

A>B .... **A** é maior que **B**
A<B .... **A** é menor que **B**
A>=B .... **A** é maior ou igual a **B**
A<=B .... **A** é menor ou igual a **B**
A=B .... **A** é igual a **B**
A<>B .... **A** não é igual a **B**

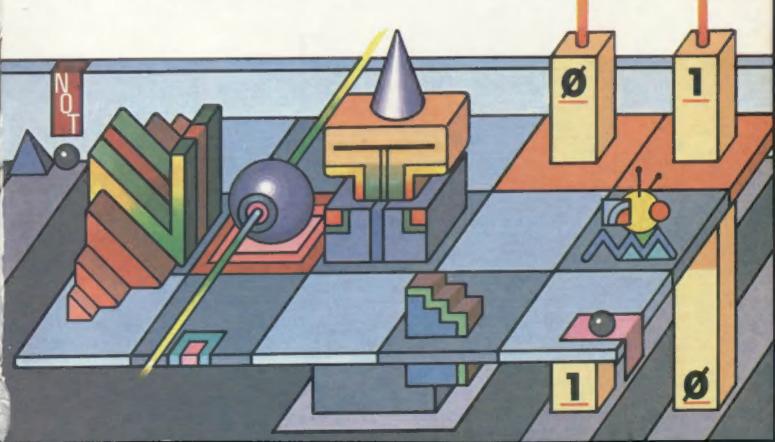
Você já deve ter visto esses operadores em vários programas de INPUT. Embora possam ser usadas em aritmética direta, é no uso conjunto com o **IF...THEN** que se torna mais evidente sua contribuição nos testes condicionais. Um exemplo comum:

```
IF A>B THAN PRINT" A É MAIOR
QUE B"
```

Neste caso, o valor de **A** deve ser maior que o de **B** para que o restante da linha seja executado. Sempre que o valor de **A** for menor que o de **B**, o programa saltará para a próxima linha. O exemplo torna evidente a possibilidade de um salto condicional: se um conjunto de valores satisfaz a condição, então o programa faz algo; se é um outro conjunto de valores que a satisfaz, então o programa faz outra coisa.

O uso dos operadores relacionais não se limita a valores numéricos. Eles também podem ser empregados para comparar cadeias. Contudo, isso requer certa cautela; caso contrário, obtêm-se resultados um tanto quanto estranhos. É importante lembrar, em primeiro lugar, que a comparação sempre pára no último caractere da cadeia mais curta. Ou seja, se uma cadeia contém onze caracteres e outra contém sete, somente os sete primeiros da cadeia mais longa serão comparados com os da cadeia mais curta.

Na realidade, a comparação é feita com um caractere de cada vez, da es-

querda para a direita — e aqui se encontra a explicação para o surgimento de resultados estranhos. Numa comparação numérica, a afirmação **5>10** é obviamente falsa mas, numa comparação entre cadeias, pode-se ter uma afirmação na forma **A$>B$**. Se **A$="5"** e **B$="10"**, o computador compara primeiro os caracteres à esquerda — **5** e **1**. Em seguida pára, pois para ele não há mais nada a ser comparado.

Assim, temos uma condição "verdadeiro" incorreta, pois **5** é maior que **1**, mas o computador ignorou o caractere que restou na cadeia mais longa. Esteja sempre atento a erros como este.

Nas cadeias, as letras do alfabeto seguem a seguinte ordem de comparação: **"A"<"B"<"C"<"D"**, e assim por diante. Mais uma vez, seja cauteloso, pois o computador não entende o significado dos caracteres. Na verdade, o que ele faz é comparar os códigos dos caracteres, o que explicaremos com mais detalhes num próximo artigo. Em conseqüência, os espaços e outros caracteres que não são letras tornam-se tão importantes quanto as próprias letras, pois cada um tem seu código. Esquecer-se disso resultaria em erro, por exemplo, num programa que coloca cadeias em ordem alfabética.

Se cada cadeia tem a mesma seqüência de caracteres, a mais longa será considerada a maior, numericamente. Mas note que a cadeia mais curta é tomada como a maior numa expressão como **"ABD">"ABCD"**, pois a comparação pára quando encontra a primeira diferença — ou seja, quando os valores dos códigos de **"D"** e **"C"** são comparados. A prioridade, portanto, é parecida com aquela dos dicionários ou índices, onde "amor" vem antes de "amoroso" mas depois de "amargo".

VERDADEIRO OU FALSO

Depois de qualquer comparação, obtém-se um resultado — um número inteiro. Este é **0**, se a comparação for falsa, e **−1** ou **1** (dependendo da máquina), se for verdadeira.

Experimente inserir no seu microcomputador, em modo direto (imediato), a seguinte linha:

```
PRINT 6>5 , 5>7
```

A expressão da esquerda é verdadeira e a da direita, falsa. Você verá aparecer na tela, portanto, o resultado **1** (ou **−1**) e **0**.

Os resultados obtidos numa comparação podem ser usados em cálculos no decorrer do programa. Mas tome cuidado com a divisão: qualquer tentativa de dividir um número por **0** resultaria numa mensagem de erro.

OPERADORES LÓGICOS

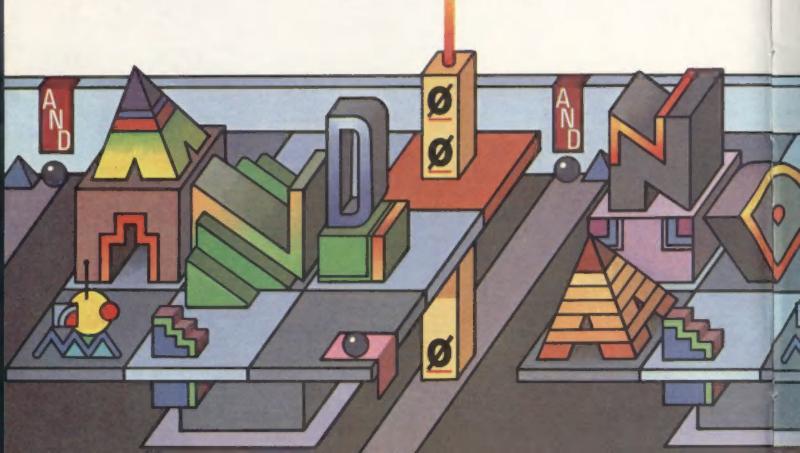
AND, **OR** e **NOT** são operadores lógicos que auxiliam na tarefa de decisão do **IF...THEN** (página 41). Por exemplo: **IF** (se) isto é verdadeiro **AND** (e) aquilo é verdadeiro **THEN** (então) faça alguma coisa. Os outros dois operadores seriam usados da mesma maneira.

Estes operadores — às vezes chamados de *operadores booleanos* (inventados pelo matemático inglês George Boole) — podem ser empregados para comparar números ou cadeias, tanto no modo direto como no modo de programa, e obter um "valor verdade" para o que se considera uma condição de teste muito complexa. Eles oferecem um caminho mais curto para o cumprimento de uma tarefa que envolveria um amontoado de **IF...THEN**.

O USO DO AND

Simplificadamente, o operador **AND** pode ser considerado como tendo o significado de **E**. Numa expressão como:

```
IF V>0 AND V<100
PRINT" PERMITIDO "
```

...a mensagem aparece na tela somente se **V** for maior que **0** e menor que **100**.

Num programa, teríamos algo como:

```
990 OKAY=1 AND MES>5 AND MES<10
 AND ANO>1900 AND ANO<2001
```

```
990 OKAY=-1 AND MES>5 AND MES<1
0 AND ANO>1900 AND ANO<2001
```

Uma linha de programa como esta poderia ser empregada para vários fins — por exemplo, para verificar se uma certa data caiu no inverno, no século vinte.

Utiliza-se o **AND**, no caso, para comandar quatro testes. Todos devem ser verdadeiros para que a variável **OKAY** permaneça como verdadeira. Nesse teste de validade, primeiramente ajusta-se a variável **OKAY** como "verdadeira" (−1, ou 1 no Sinclair e Apple). Depois, a condição necessária é que **MES** esteja entre **6** e **9** e que **AND** esteja entre **1901** e **2000** (inclusive).

Verifiquemos mais de perto o que acontece: se as condições forem totalmente satisfeitas, todas as expressões produzirão um valor verdadeiro. Em conseqüência, a linha de programa ficaria assim:

```
990 OKAY= -1 AND -1 AND -1 AND
-1 AND -1
```

(nos micros da linha Sinclair e Apple seria 1, em vez de −1)

Se acrescentarmos **PRINT OKAY**, verificaremos que, de fato, o resultado é −1, ou seja, verdadeiro.

O USO DO OR

Embora o significado de **OR** (traduzindo, teríamos **OU**) seja diferente do de **AND**, ambos podem ser considerados semelhantes em relação à maneira como são usados.

Vejamos um exemplo simples, que emprega **OR** na comparação de valores de uma determinada variável.

```
IF V=8 OR V=10 PRINT" OKAY "
```

A mensagem aparecerá na tela se o valor de **V** for **8** ou **10**.

O emprego do **OR** é muito útil na verificação da validade dos dados introduzidos no computador via comando **INPUT**. Se temos, por exemplo, um programa que pergunta pela idade, com certeza haverá alguém que, só por curiosidade, responderá −10 ou 999.

Para contornar problemas como esse, usamos a seguinte alternativa:

```
10 INPUT A
20 IF A<1 OR A>120 THEN PRINT"
   SEJA SENSATO ": GOTO 10
```

O USO DO NOT

O terceiro operador lógico de uso bastante comum é o **NOT**. Ele difere um pouco dos outros, pois age somente na expressão numérica ou lógica que o segue — **AND** e **OR** comparam expressões nos dois lados, mas **NOT** trabalha sobre um único valor.

Veja um exemplo do uso do **NOT**.

```
IF NOT (A>10) THEN 999
```

Se traduzíssemos isto para uma linguagem do dia-a-dia teríamos: "se o valor de **A** não é maior que **10**, então vá para a linha **999**".

A função lógica do **NOT** é converter para falsa uma condição verdadeira, ou vice-versa. Poderíamos utilizá-lo, por exemplo, como uma chave que inverte a condição falso/verdadeiro obtida no resultado de uma operação anterior.

TABELAS - VERDADE

Ouvimos falar de "tabela-verdade" sempre que o assunto é operadores lógicos. Bem, elas são muito simples. Sua

função consiste em fornecer uma representação visual do que acontece com os falso/verdadeiro quando usamos **AND** e **OR** sobre eles. **NOT** não precisa de tabela, já que produz sempre o inverso.

As tabelas podem assumir várias formas; uma típica para o **AND** é:

A	B	C	
−1	−1	−1	linha 1
−1	0	0	linha 2
0	−1	0	linha 3
0	0	0	linha 4

Para desvendar o significado da tabela, basta interpretá-la linha por linha. Linha 1: "Se A é verdadeiro e B é verdadeiro, então C também é verdadeiro". Linha 2: "Se A é verdadeiro e B é falso, então C é falso". Linha 3: "Se A é falso e B é verdadeiro, então C é falso". Linha 4: "Se A e B são ambos falsos, então C é falso".

A tabela-verdade para o **OR** é:

A	B	C	
−1	−1	−1	linha 1
−1	0	−1	linha 2
0	−1	−1	linha 3
0	0	0	linha 4

Na primeira linha, lê-se: "Se A é verdadeiro e B é verdadeiro, então C é verdadeiro". Nas linhas 2 e 3 lê-se: "Se A ou B for verdadeiro, então C é verdadeiro". A linha 4 significa: "Se A e B são ambos falsos, então C também é falso"

STT

Apresentamos aqui um programa que usa **AND**, **OR** e **NOT**. Trata-se de uma versão simplificada de um programa que calcula a escala de salário correta para o candidato a um emprego.

```
10 INPUT"IDADE";IDADE
20 INPUT"NUMERO DE REFERENCIAS"
:REF
30 MAIOR18=(IDADE>=18)
40 QUAL=(REF>=5)
50 IF NOT MAIOR18 AND NOT QUAL
THEN PRINT "NAO QUALIFICADA"
60 IF(NOT MAIOR18 AND QUAL)OR(M
AIOR18  AND NOT QUAL) THEN PRIN
T "ESCALA 1 DE SALARIO"
70 IF MAIOR18  AND QUAL THEN PR
INT "ESCALA 2 DE SALARIO"
```

Digamos que o candidato tenha respondido **20** para idade e **4** para número de referências. Isso significa que **(IDADE> =18)** é verdadeiro, mas **(REF> =5)** é falso. A pessoa, portanto, é **MAIOR18** e **NÃO QUAL**ificada. Na linha 60, encontramos outro conjunto de condições que, caso satisfeitas, seleciona a primeira escala de salários. Poderíamos facilmente modificar o programa para testar mais condições, por exemplo: verificar se a pessoa tem mais de dois anos de experiência, etc.

STT

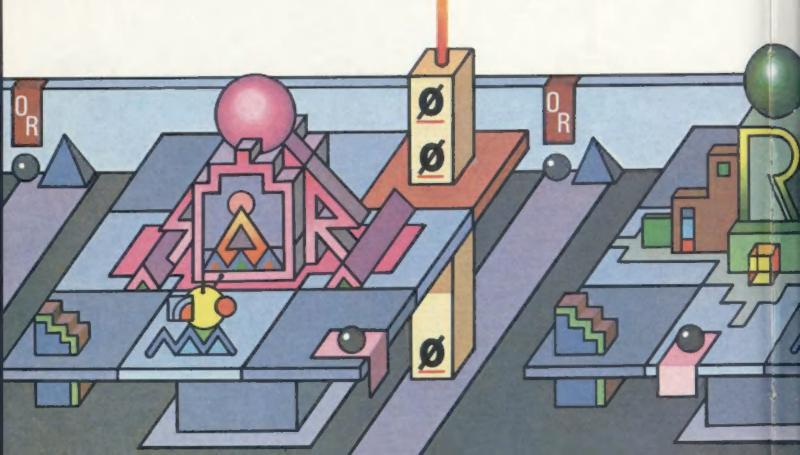
OPERAÇÕES NUMÉRICAS

O uso dos operadores lógicos não está limitado somente ao **IF...THEN**. Eles também podem ser usados com alguns tipos de operações numéricas. Talvez você tenha visto esse tipo de uso em programas anteriores, mas ficou sem saber o que estava acontecendo. Na verdade, eles nem sempre funcionam da maneira óbvia, como poderíamos esperar. Tente isto no modo direto:

```
PRINT 375 AND 47
```

Se você esperou que o resultado fosse 422 ou até mesmo 37547, errou. Na verdade, temos 39 como resultado. O que estaria acontecendo então? Somente se nos aprofundarmos um pouco no funcionamento dos computadores poderemos entender o que ocorre com o **AND** desse exemplo.

Em primeiro lugar, as expressões (375 e 47) são transformadas em números in-

teiros de dois bytes. Na forma binária, ficam assim:

375 em binário de dois bytes:
0000000101110111
47 em binários de dois bytes:
0000000000101111

Em seguida, o **AND** compara bit a bit todos os dezesseis bits, produzindo um "1" somente quando, no par de bits comparados, ambos forem "1". Da direita para a esquerda, os únicos pares de bits que satisfazem (produzem "1") são os de números 0,1,2 e 5. Isto resulta no número binário 0000000000100111 que, transformado para decimal, é 39. Portanto, 375 **AND** 47 é igual a 39.

Agora, tente em modo direto:

```
PRINT 375 OR 47
```

Como obtivemos 383? Lembre-se da propriedade do **OR**: produz-se "1" quando, no par de bits comparados, pelo menos um dos bits for "1". Observando novamente a forma binária de 375 e 47, vemos que os pares de bits de número 8, 6, 5, 4, 3, 2, 1 e 0 produzem "1" quando comparados pelo **OR**. Isto resulta no binário 0000000101111111, que equivale ao 383 decimal.

Com **OR** é possível obter o mesmo resultado, usando expressões diferentes. **PRINT 375 OR 75**, por exemplo, também resulta em 383. Caso queira verificar, faça o seguinte: converta para binário, compare pelo **OR** e converta de volta para decimal o resultado obtido.

O valor −1 (armazenado como FFFFFFFF em hexadecimal — um arranjo de bits onde todos valem 1) não se altera quando comparado pelo **OR** com qualquer outro valor. Faça uma experiência, tentando no modo direto:

```
PRINT -1 OR 375
```

O resultado é −1.

Vejamos agora uma aplicação numérica para o **NOT**.

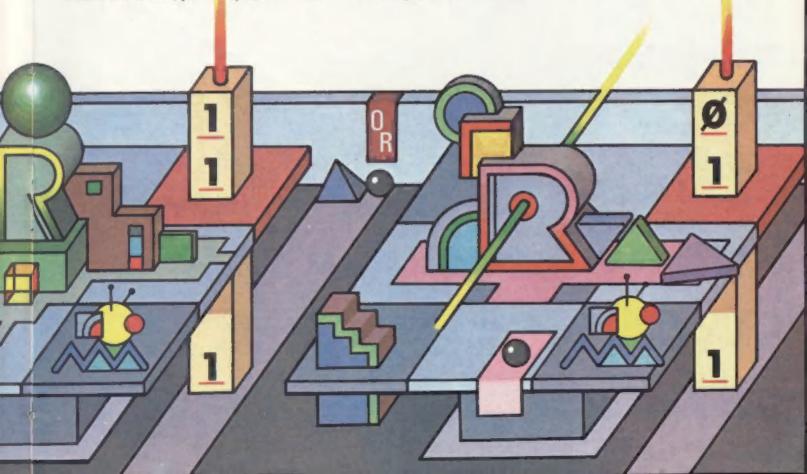
```
PRINT NOT 10.75 , NOT -11
```

O resultado é −11 e 10. O **NOT** somou 1 ao valor e depois mudou seu sinal. Note que a parte não inteira (.75) foi ignorada e eliminada, e que o novo valor pode ser estimado usando $X = -(X+1)$. Na realidade, o valor é transformado num inteiro de quatro bytes, com todos seus bits invertidos.

Os operadores lógicos destes computadores não fazem comparação bit a bit dos números e, por isso, quase nunca são usados em operações numéricas.

EXISTE LIMITE PARA O TAMANHO DOS NÚMEROS USADOS EM *AND* E *OR*?

No TRS-Color e MSX, os números empregados em qualquer operação com **AND** ou **OR** têm que estar entre −32768 e +32767; caso contrário, teremos uma mensagem de erro.

No Spectrum, Apple e TK-2000 não é possível usar números nas operações **AND** ou **OR**.

OS OBJETOS DA AVENTURA

Chegou a hora de preencher o mundo ainda vazio de nossa aventura. Você verá aqui como acrescentar ao programa uma lista de objetos e, também, como manipulá-los.

Na última seção de Programação de Jogos, dispúnhamos de um conjunto completo de posições para nossa aventura e o jogador já estava capacitado a explorar todas elas. As movimentações do aventureiro, porém, até agora não têm propósito, uma vez que o mundo da aventura está vazio. Convém, assim, retomar o planejamento inicial e examinar o que se pretendia incluir em cada ponto.

Veremos, em seguida, como adicionar ao programa as rotinas que colocarão os objetos envolvidos na aventura em seus devidos lugares. Outras rotinas permitirão ao jogador carregar objetos consigo ou deixá-los de lado. Escreveremos, ainda, uma rotina para listar um inventário de todos os itens utilizados — ela será importante para o jogador, quando ele se deparar com um problema e precisar verificar exatamente a situação de cada objeto.

Use o **LOAD** e carregue o programa usado recentemente, deixando-o pronto para receber as novas rotinas.

OBJETOS

A máquina precisa saber três coisas sobre cada objeto da aventura: o número de posição onde foi inicialmente colocado, seu nome (ou descrição curta) e, finalmente, a descrição detalhada do local onde se encontra. As três informações são necessárias já que, primeiro, o computador deverá escolher um objeto adequado a cada posição. Depois, precisará contar ao aventureiro que objetos estão nesta posição — mesmo que use uma longa descrição. E, por último, terá que dispor de um nome para usar em instruções e na lista.

Os números de posição serão colocados em uma matriz, o nome do objeto em outra, e a descrição longa em uma terceira. As três matrizes serão manipuladas em paralelo pelo programa — cada elemento da matriz guardará um espaço equivalente de informação sobre o objeto: o primeiro conterá o número da posição, o segundo o nome do objeto e assim por diante. Adicione estas linhas ao seu programa:

- LINHAS **DATA** PARA A LISTA DE OBJETOS
- DESCRIÇÕES CURTAS E LONGAS
- COLOCAÇÃO DOS OBJETOS NAS POSIÇÕES CORRETAS
- MAIS VERBOS
- COMO PEGAR E LARGAR OBJETOS
- COMO MOSTRAR AO JOGADOR A LISTA DOS OBJETOS QUE CARREGA

S

```
160 REM **PREPARA MATRIZES DE
OBJETOS**
170 READ NB
180 DIM B(NB): DIM B$(NB,14):
DIM S$(NB,40)
190 FOR I=1 TO NB: READ B(I),B
$(I),S$(I): NEXT I
200 DATA 7,4,"SACO","HA UM SAC
O DE BOLAS DE GUDE AQUI"
210 DATA 14,"TIJOLO","TEM UM T
IJOLO NO CHAO"
220 DATA 24,"CORRENTE","HA UMA
 CORRENTE PENDURADA SOBRE O TR
ONO"
230 DATA 0,"REVOLVER","TEM UM
REVOLVER NO CHAO"
240 DATA 0,"OLHO","UM OLHO CRA
VEJADO DE BRILHANTES ESTA NO C
HAO"
250 DATA 22,"LAMPADA","VOCE ES
TA DIANTE DE UMA LAMPADA"
260 DATA 0,"COLETOR","DE REPEN
TE SURGE UM COLETOR DE IMPOSTO
S"
```

T T /X/ [1] [6]

```
160 REM **PREPARA MATRIZES DE O
BJETOS**
170 READ NB
180 DIM OB(NB),OB$(NB),SI$(NB)
190 FOR I=1 TO NB:READ OB(I),OB
$(I),SI$(I):NEXT
200 DATA 7,4,SACO,HA UM SACO DE
 BOLAS DE GUDE AQUI
210 DATA 14,TIJOLO,TEM UM TIJOL
O NO CHAO
220 DATA 24,CORRENTE,HA UMA COR
RENTE PENDURADA SOBRE O TRONO
230 DATA 0,REVOLVER,TEM UM REVO
LVER NO CHAO
240 DATA 0,OLHO,UM OLHO CRAVEJA
DO DE BRILHANTES ESTA NO CHAO
250 DATA 22,LAMPADA,VOCE ESTA D
IANTE DE UMA LAMPADA
260 DATA 0,COLETOR,DE REPENTE S
URGE UM COLETOR DE IMPOSTOS
```

Cada linha entre 200 e 260 contém três partes dos dados referentes ao mesmo objeto. A linha 200 apresenta um dado a mais em sua lista. O número 7 — o primeiro do comando **DATA** — diz à máquina quantos conjuntos de dados existem.

Uma vez que o número 7 tenha sido

lido pela linha 170, três matrizes serão dimensionadas para este tamanho, pela linha 180. **OB** conterá a posição de cada objeto — um número da posição, ou 0, se ■ elemento ainda não existe (é como se fosse ■ tampa de um baú, que precisasse ser aberto a cada aventura), e −1, se está sendo levado pelo aventureiro. **OB$** conterá a descrição curta e **SI$** a descrição longa.

A linha 190 completará a matriz com os dados das linhas 200 a 260. Os comandos **DATA** estão arrumados em conjuntos de três elementos: um número de posição, a descrição curta do objeto ■ a descrição longa do objeto.

Para [illegible] a mesma rotina em outras aventuras, não é necessário fazer muitas alterações nessa estrutura, pois, ajustando a primeira parte dos dados, automaticamente os comandos **FOR... NEXT** e ■ dimensões das matrizes estarão ajustados.

[illegible] POSICIONAR OS [illegible]

O programa contém agora todas as informações sobre os objetos e as posições onde deverão ser colocados. A rotina seguinte exibe a descrição longa do objeto quando o jogador está no local adequado.

```
360 REM **COLOCA CADA OBJETO E
M SEU LUGAR**
370 [illegible] I=1 TO NB: IF B(I)=L
THEN  PRINT S$(I)
380 NEXT I
```

```
360 REM**COLOCA CADA OBJETO EM
SEU LUGAR**
370 FOR  I=1 TO NB:IF OB(I)=L T
HEN PRINT SI$(I)
380 NEXT
```

Neste estágio, faça uma pequena alteração nas linhas 330 e 340: troque **GOTO 400** para **GOTO 370**. As linhas 370 e 380 checam a matriz que guarda a posição do objeto. Se algum número de posição combinar com a posição corrente — L — uma descrição curta será exibida após a descrição da posição. Essa rotina pode ser usada em outras aventuras, sem alterações.

MAIS VERBOS

A aventura inclui objetos em diferentes posições mas, como a máquina até aqui não entende qualquer palavra ex-

ceto NORTE, SUL, LESTE e OESTE, o pobre aventureiro não pode fazer nada com eles. Imagine sua frustração ao se ver incapaz de pegar o tão desejado saquinho de bolas de gude, ou sem condições de se defender do fiscal da Receita! Precisamos fornecer ao computador um vocabulário de palavras que ele possa reconhecer, informando o que fazer com os objetos. Mais tarde, veremos como proceder ■ o jogador entrar uma palavra que não está no vocabulário programado.

Como o programa trata todas as palavras de direção como verbos, o melhor lugar para os verbos, que descrevem o que fazer com os objetos, é a matriz **R$** e, para os números correspondentes, **R**.

Precisaremos, no entanto, fazer algumas alterações na linha 130. Os limites do **FOR...NEXT** deverão ser mudados. Reescreva toda a linha ou use o editor da máquina para mudá-la. Seja qual for sua escolha, a linha 130 será agora:

```
130 FOR K=1 TO 19: READ R$(K),
R(K): NEXT K
```

```
130 FOR K=1 TO 19:READ R$(K),R(
K):NEXT
```

Agora, adicione as linhas 140 e 145:

```
140 DATA "NADAR","5","ESVAZIAR
","6","ACENDER","7","DESISTIR"
,"8","LISTAR","9","MATAR","10"
,"ATIRAR","10","AJUDAR","11"
145 DATA "PEGAR","2","APANHAR"
,"2","CARREGAR","2","COLOCAR",
"3","DEIXAR","3","LARGAR","3",
"PUXAR","4"
```

```
140 DATA NADAR,5,ESVAZIAR,6,ACE
NDER,7,DESISTIR,8,LISTAR,9,MATA
R,10,ATIRAR,10,AJUDAR,11
145 DATA PEGAR,2,APANHAR,2,CARR
EGAR,2,COLOCAR,3,DEIXAR,3,LARGA
R,3,PUXAR,4
```

A cada verbo corresponde um número. Verbos com ■ mesmo número possuem o mesmo significado e, portanto, o mesmo efeito. Planejamos o programa de modo que o computador reconheça, por exemplo, **PEGAR**, **APANHAR** e **CARREGAR**, evitando que o aventureiro gaste seu precioso tempo tentando descobrir qual desses verbos usar. Você pode facilmente adicionar suas próprias palavras às linhas de comando **DATA**: basta trocar o laço **FOR...NEXT** na linha 130 ■ colocar outro comando **DATA** após a linha 145. Você deverá fazer algumas alterações em outros locais do programa mas, nas próximas seções de Programação de Jogos, informaremos exatamente como proceder.

... DAS ROTINAS ADEQUADAS

Depois de entrar todos os verbos na última rotina, o computador precisará de outras rotinas que o tornem capaz de agir conforme as instruções e de fazer com que o aventureiro carregue alguns objetos.

A sub-rotina que começa na linha 3010, por exemplo, define **V$**, **N$** e **I** (um número da matriz **R** que você já deve ter escrito).

Esta pequena rotina fará com que a máquina seja capaz de selecionar a rotina correta, de acordo com o valor de **I** — que é o significado da entrada do aventureiro.

O Spectrum não tem o comando **ON...GOTO**, usado em programas para outros computadores. As linhas do programa, portanto, têm que ser um pouco diferentes.

Já temos uma matriz, **G**, que contém números de linha para as descrições de posição. Os números de linha necessários às novas rotinas podem ser adicionados a esta matriz.

Por isso, a linha 30 ficou assim:

```
30 FOR N=1 TO 4: FOR M=1 TO
11: READ G(M,N): NEXT M: NEXT
 N
```

E é também pelo motivo acima que essa linha contém todos os números de linha que precisaremos (veja a explicação completa no artigo da página 270).

```
70 DATA 1010,1150,1240,1310,
1410,1460,1500,1360,1080,1550
,3110
```

Agora, adicione a rotina que solucionará a rotina correta, de acordo com o valor de **I**:

```
500 REM **SELECIONA OPCAO**
510 IF I=0 THEN  GOTO 520
520 PRINT '"EU NAO SEI COMO ";
V$: GOTO 370
```

Se a sub-rotina de verificação de instrução que começa na linha 3010 não encontrar uma combinação para **V$** em **R$**, **I** torna-se zero e a linha 510 faz com que ■ mensagem "EU NÃO SEI COMO" surja na tela. Se **I** tiver qualquer outro valor, a linha 515 encontra o número de linha correto na matriz **G** e executa um **GOTO**.

```
500 REM**SELECIONA OPCAO**
505 IF I=0 THEN GOTO 520
510 ON I GOTO 1010,1150,1240,13
10,1410,1460,1500,1360,1080,155
0,3110
520 PRINT:PRINT "EU NAO SEI COM
O ";V$:GOTO 370
```

Os números após o **ON...GOTO** na linha 510 iniciam as diversas rotinas. Cada valor de **I** é um verbo diferente ou um grupo de verbos. Se **I = 10**, por exemplo, a rotina "MATAR" será selecionada — e, sendo o décimo número na linha, ela começará na linha 1550.

Se a sub-rotina de verificação de instrução que começa na linha 3010 não encontrar uma combinação de **V$** em **R$**, **I** torna-se 0. Nesse caso, o **ON...GOTO** na linha 510 não terá nenhum efeito. A mensagem na linha 520 será exibida na tela.

COMO PEGAR OS OBJETOS

Já temos uma rotina para quando **I** for igual ■ 1, o que ocorre quando o aventureiro dá uma ordem. Ela está entre as linhas 1010 e 1060.

Quando **I = 2**, o aventureiro digitou uma rotina "PEGAR" — ou seja, as palavras **PEGAR**, **APANHAR** ou **CARREGAR**. Esta rotina, apresentada a seguir, permitirá ao aventureiro pegar e conservar consigo qualquer objeto que esteja na posição.

```
1140 REM **PEGAR**
1150 FOR G=1 TO NB
1160 IF N$=B$(G, TO LEN N$) THE
N  GOTO 1190
1170 NEXT G
1180 PRINT N$;"???": GOTO 330
1190 IF B(G)=-1 THEN  PRINT "VO
CE PEGOU": GOTO 330
1200 IF B(G)<>L THEN  PRINT "NA
O ESTA AQUI": GOTO 330
1210 PRINT "OK": LET B(G)=-1
1220 GOTO 330
```

```
1140 REM**PEGAR**
1150 FOR G=1 TO NB
1160 IF INSTR(OB$(G),N$)=1 THEN
 GOTO 1190
```

```
1170 NEXT
1180 PRINT N$;"???":GOTO 330
1190 IF OB(G)=-1 THEN PRINT "VO
CE PEGOU":GOTO 330
1200 IF OB(G)<>L THEN PRINT "NA
O ESTA AQUI":GOTO 330
1210 PRINT "OK":OB(G)=-1
1220 GOTO 330
```

```
1140   REM ** PEGAR **
1150   FOR G = 1 TO NB
1160   IF N$ = LEFT$ (OB$(G), L
EN (N$)) THEN 1190
1170   NEXT
1180   PRINT N$;" ??"; GOTO 330
1190   IF OB(G) =  - 1 THEN  PRI
NT "VOCE PEGOU": GOTO 330
1200   IF OB(G) <  > L THEN  PRI
NT "NAO ESTA AQUI": GOTO 330
1210   PRINT "OK":OB(G) =  - 1
1220   GOTO 330
```

As linhas 1150 a 1170 procuram a matriz contendo a descrição curta — **B$**, no caso do Spectrum, e **OB$** nos demais computadores — do objeto que o aventureiro chamou. Se este é encontrado, o programa pula para a linha 1190. Caso contrário, a linha 1180 exibe o nome do objeto que o aventureiro digitou, seguido de pontos de interrogação.

Depois de encontrar o nome do objeto, duas verificações são feitas. A linha 1190 checa o elemento da matriz de posição do objeto — **B** ou **OB** —, para saber se ele já foi levado. Em caso afirmativo (o valor do elemento da matriz é −1), a mensagem "VOCÊ PEGOU" será exibida.

A linha 1200 verifica se o objeto está presente, examinando novamente a posição da matriz. Se ele não estiver presente, o programa exibirá a mensagem "NÃO ESTÁ AQUI". Você poderá trocar essa mensagem, se ela não servir para a sua aventura.

Caso o objeto não tenha sido levado e se encontre na mesma posição que o aventureiro, a linha 1210 exibirá "OK" e o elemento na matriz de posição de objetos será mudado para −1.

COMO DEIXAR OBJETOS DE LADO

A rotina "DEIXAR" faz o contrário. Ela permite que o jogador abandone qualquer dos objetos que pegou.

```
1230 REM **DEIXAR**
1240 FOR G=1 TO NB
1250 IF N$=B$(G, TO LEN N$) THE
N  GOTO 1270
1260 NEXT G: PRINT N$;"???": GO
TO 330
1270 IF B(G)<>-1 THEN  PRINT "V
OCE NAO PODE LARGAR O QUE NAO T
EM": GOTO 330
1280 PRINT "OK": LET B(G)=L
1290 GOTO 330
```

```
1230 REM**DEIXAR**
1240 FOR G=1 TO NB
1250 IF INSTR(OB$(G),N$)=1 THEN
 1270
1260 NEXT:PRINT N$:"???":GOTO 3
30
1270 IF OB(G)<>-1 THEN PRINT "V
OCE NAO PODE LARGAR O QUE NAO T
EM":GOTO 330
1280 PRINT "OK":OB(G)=L
1290 GOTO 330
```

```
1230   REM ** DEIXAR **
1240   FOR G = 1 TO NB
1250   IF N$ = LEFT$ (OB$(G), L
[illegible] (N$)) THEN 1270
1260   NEXT : PRINT N$;" ??": GO
TO 330
1270   IF OB(G) <  >  - 1 THEN
PRINT "VOCE NAO PODE LARGAR O [illegible]
UE NAO TEM": GOTO 330
1280   PRINT "OK":OB(G) = L
1290   GOTO 330
```

Seu funcionamento se assemelha muito ao da rotina "PEGAR", vista anteriormente. A matriz de descrição curta é mais uma vez procurada — agora nas linhas 1240 a 1260. Se o objeto pedido estiver na matriz, a linha 1270 verifica se está sendo levado pelo aventureiro. Se não estiver, o computador exibirá a mensagem "VOCÊ NÃO PODE LARGAR O QUE NÃO TEM".

Se o aventureiro estiver carregando o objeto, a linha 1280 exibe "OK" e o elemento apropriado na matriz de posição de objetos — **OB** ou **B** — é ajustado. Ele toma, então, o número da posição corrente — L —, já que −1 significa que o objeto estava sendo carregado.

A LISTAGEM DO SAQUE

O aventureiro distraído ficará muito grato ao obter uma lista de todos os objetos que carrega.

Aqui está uma rotina que faz exatamente isso:

```
1070 [illegible] **LISTAR**
1080 PRINT "VOCE [illegible] ";: LET IN
=0
1090 FOR G=1 TO [illegible]
1100 IF B(G)=-1 THEN  PRINT TAB
 10;B$(G): LET IN=IN+1
1110 NEXT G
1120 IF IN=0 THEN  PRINT "NADA"
1130 GOTO 330
```

```
1070 REM**LISTAR**
1080 PRINT "VOCE TEM ";:IN=0
1090 FOR G=1 TO NB
1100 IF OB(G)=-1 THEN PRINT TAB
(10)OB$(G):IN=IN+1
1110 NEXT
1120 IF IN=0 THEN PRINT "NADA"
1130 GOTO 330
```

```
1070   REM ** LISTAR **
1080   PRINT "VOCE TEM ";:IN = 0
1090   FOR G = 1 TO NB
1100   IF OB(G) =  - 1 THEN  PRI
NT  TAB( 10);OB$(G):IN = [illegible] + 1
1110   NEXT
1120   IF IN = 0 THEN  PRINT "NA
DA"
1130   GOTO 330
```

"VOCÊ TEM" será exibido pela linha 1080, antes que se inicie a listagem dos objetos. O laço **FOR...NEXT** checa, um a um, os elementos da matriz de posições de objetos. Dessa vez, os elementos importantes são os que contêm −1, indicando que o objeto está sendo carregado. Se o valor de qualquer um dos elementos for −1, a descrição curta do objeto será, então, exibida. O contador do inventário I será incrementado de 1.

Se nenhum objeto estiver sendo carregado, o contador **IN** continua em zero e a linha 1120 exibe "NADA", em vez da lista de objetos.

As rotinas "PEGAR", "DEIXAR" e "LISTAR" podem ser usadas como estão, se **NB** for definido em uma rotina anterior.

Agora, o programa está pronto para receber as rotinas finais, que serão apresentadas num próximo artigo. Elas se referem ao fiscal da Receita, ao tijolo, à lâmpada, à procura do globo ocular, ao término da aventura e, finalmente, à instrução descrevendo o objeto da busca.

Se você fizer uma experiência, executando o programa neste estágio, verá que, enquanto parte dele funciona, alguma coisa estranha acontece. A razão para isso é que o programa requer um número de rotinas que ainda não existe. Se você entrar certas palavras, o programa tentará desviá-las para linhas inexistentes.

COMO EVITAR E DETECTAR ERROS

- MENSAGENS DE ERROS
- ORIENTAÇÕES OBSCURAS
- OS ERROS MAIS COMUNS
- TESTE SUA HABILIDADE EM DETECTAR ERROS

Nenhum programa está livre de erros. Aprenda a localizá-los e habitue-se a corrigi-los na medida em que aparecem. Esta é, sem dúvida, a melhor maneira de evitar problemas mais sérios.

Nenhum programa, de qualquer tamanho, que esteja sendo desenvolvido ou simplesmente copiado de uma listagem, está completamente livre de erros — os "grilos". Estes aparecem pelas mais diversas razões e comprometem o programa em graus diferentes.

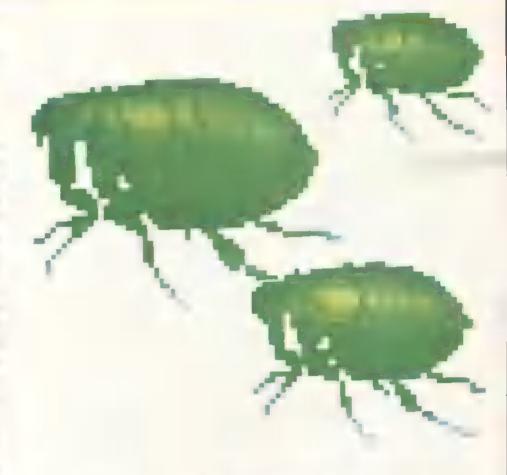
Os erros mais sérios podem impedir que um programa seja rodado. Outros, simplesmente, permanecem ocultos, até que certa rotina ou seqüência de entradas os revele. Por exemplo, em um jogo de aventuras, tudo pode funcionar perfeitamente até que o jogador tome um determinado caminho, carregando uma faca — e cai num buraco que não deveria estar ali. Ou, em um programa de contabilidade, pode-se de repente encontrar um erro enorme, que afeta apenas as entradas feitas na segunda semana de dezembro.

O primeiro tipo de erro deve ser completamente eliminado antes da utilização do programa.

E o segundo? Bem, o ideal seria procurá-los sistematicamente, para evitar surpresas. A melhor maneira de fazê-lo é testar cuidadosamente o programa — ■ que inclui tentar o inesperado, como entrar uma seqüência "impossível" de entradas. Para auxiliar nesse tipo de tarefa, existem rotinas muito úteis, especiais para detecção de erros. Logo falaremos sobre elas.

MENSAGENS DE ERRO

Todos os computadores domésticos produzem mensagens de erro, de um tipo ou de outro. Você já deve ter encontrado várias delas, desde que começou a utilizar o computador.

Tais mensagens variam de códigos simples de números e letras ■ descrições completas que deixam poucas dúvidas sobre a natureza do erro — ainda que, às vezes, não apontem diretamente o próprio erro.

Maiores detalhes sobre as mensagens de erro podem ser encontrados no manual do seu computador. Consulte-o sempre que não compreender ■ significado exato de alguma delas. Só depois comece ■ trabalhar na correção do erro.

"DEPURE" O PROGRAMA

Para evitar problemas maiores, é muito importante localizar e corrigir cada erro na medida em que aparece. Não deixe um erro permanecer em um programa, mesmo que este pareça rodar satisfatoriamente.

Caso contrário, sempre haverá uma chance de que o erro apareça mais tarde, numa hora crítica. A conseqüência poderá ■■ um desastre para o programa, o que resultará em muita digitação extra, ou — pior ainda — na perda dos dados disponíveis, se o programa for colocado em uso efetivo. No final, você não terá mais a oportunidade de resolver o problema.

Corrigir os erros, ou seja, "depurar" o programa, pode se tornar uma tarefa terrivelmente complicada se você não se empenhar em isolar cada problema de modo sistemático. Ao digitar um programa pronto ou ao desenvolver o seu próprio, é muito provável que cometa mais de um erro. Trate cada um individualmente — ou seja, localize e corrija o primeiro deles e só depois se dirija ao seguinte.

TABELA DE LOCALIZAÇÃO

Esta é uma linha de mensagens de erros e comunicados para cada computador. Quando uma entrada for precedida por um asterisco (*), o erro provavelmente se encontra na linha cujo número está indicado na mensagem ou, então, é o resultado imediato de uma entrada direta ou de uma atividade (tal com SAVE). Sempre existem exceções, especialmente quando uma variável, que é a causa de um erro em determinada linha, tem seu valor designado em uma linha previamente executada.

Quando digitar os programas, seja cuidadoso ao incluir espaços.

*NEXT sem FOR, variável inexistente, Erro de índice, *Memória lotada, *Excede área de vídeo, Número excede limite, RETURN sem GOSUB, *Fim de arquivo, *Stop executado, Argumento inválido, Inteiro excede limite, *Erro de sintaxe, *BREAK-CONT repete, Fim de dados, *Nome inválido, *Sem memória disponível, STOP em INPUT, FOR sem NEXT, Periférico inválido, *Cor inválida, BREAK-CONT prossegue, RAMTOP válido, *Comando perdido, *Canal inválido, FN sem DEF, Erro de parâmetro, *Erro de leitura.*

*/O, AO, BS, *CN, *DD, *DN, DS, FC, FD, FM, *ID, IE, *IO, LS, NF, NO, OD, OM, OS, OV, RG, *SN, *ST, *TM, *UL.*

*"CONT" ILEGAL, *DIVISÃO POR ZERO, *DIRETO ILEGAL, *VALOR ILEGAL, "NEXT" SEM "FOR", FIM DE DATA, FALTA MEMÓRIA, *FÓRMULA COMPLEXA, *S/ESPAÇO, *REDIMENSIONAR, "RETURN" SEM "GOSUB", "STRING" LONGA, ÍNDICE ILEGAL, *SINTAX ERRO, *TIPO INCOMPAT, *LINHA INDEFINIDA, FUNÇÃO INDEF.*

*"NEXT" SEM "FOR", *ERRO SINTAXE, "RETURN" SEM "GOSUB", SEM "DATA", FUNÇÃO ILEGAL, *OVERFLOW, FALTA MEMÓRIA, *Nº LINHA INEXISTENTE, ÍNDICE FORA DO LIMITE, "DIM" REDEFINIDO, *DIVISÃO POR ZERO, DIRETO ILEGAL, TIPO DESIGUAL, *FALTA ÁREA *"STRING", *"STRING" LONGA, *"STRING" COMPLEXA, NÃO CONTÍNUO! FUNÇÃO NÃO DEFINIDA, ERRO/PERIFÉRICO, ERRO/VERIF., "RESUME", "RESUME" SEM "ERROR", *FALTA OPERANDO, *LINHA MUITO LONGA.*

LOCALIZAÇÃO DOS ERROS

Não se esqueça de que o erro mais simples é, em geral, o mais difícil de ser isolado, e — ao contrário do que poderá pensar — quase sempre é a única causa da dificuldade em rodar um programa adequadamente.

Muitos problemas podem ser evitados com a utilização de um conjunto de rotinas para depurar os programas. Existem algumas técnicas bem simples e caminhos muito eficazes.

Para começar, muitas mensagens de erro apontam diretamente para a linha na qual ocorre o erro. Isso inclui o mais comum deles — **ERRO DE SINTAXE** —, além de vários outros (veja a tabela). Alguns, porém, são menos evidentes, e indicam erros em linhas que estão perfeitamente corretas. Você pode obter uma mensagem como **E Fim de dados 10:2** (o exemplo é do Spectrum, mas outros computadores apresentam mensagens semelhantes). Aparentemente, trata-se de uma indicação de que o erro se encontra na segunda instrução da linha 10. Na verdade, a segunda instrução da linha 10 diz ao computador para ler alguns dados, o que poderia aparecer também na linha 200 ou até mesmo na linha 2000. Por outro lado, talvez o erro estivesse nas linhas de dados, e não na linha 10.

Esse tipo de erro é, evidentemente, muito mais difícil de se localizar, já que não existe nenhuma indicação precisa de onde se encontra o verdadeiro problema.

As mensagens de erro, que aponta a linha, revelam apenas que o problema deve estar relacionado à maneira como se executou uma entrada em determinada linha do programa.

Inicie efetuando uma listagem do programa a partir de um ponto antes do número da linha sugerida na mensagem de erro. Algumas vezes, é útil listar a própria linha, separadamente e um pouco além das outras, se isso for possível em seu computador.

Antes de mais nada, verifique se você não cometeu nenhum tipo de *erro literal* — ou seja, se escreveu algo de modo errado ou se utilizou uma letra no lugar de um número (ou vice-versa). Seja especialmente cuidadoso em relação aos espaços e à pontuação. E procure ver se falta algum caractere — é muito fácil, por exemplo, esquecer um parêntese em uma seqüência que utilize vários deles. Examine com atenção as palavras-chave, letra por letra, pois também é bastante comum a inversão da ordem dos caracteres.

Os erros literais costumam ser uma causa freqüente de problemas, perturbando o desenvolvimento do trabalho tanto de iniciantes quanto de especialistas.

LINHA POR LINHA

Quando você sabe que o erro está em determinada linha, e esta contém duas instruções, precisará descobrir qual delas apresenta o problema. O melhor método para isso é colocar uma instrução **STOP** em uma nova linha, imediatamente após a que foi indicada pela mensagem de erro. Em seguida, entre uma instrução **REM** bem no início da última declaração e rode novamente o programa — se nenhum desastre tiver acontecido até este ponto, é claro. Caso o erro de sintaxe não apareça, você saberá que ele se encontra na última declaração.

Pode-se utilizar um método parecido para descobrir, entre algumas linhas, a que contém o erro. Basta inserir sucessivamente, entre cada linha, uma linha extra com a instrução **STOP**, como mais tarde explicaremos em detalhe.

O método, porém, não pode ser ampliado com segurança para as linhas com instruções múltiplas, pois qualquer coisa antes do **REM** e depois da próxi-

ma linha do programa será ignorada. Nestes casos, deve-se dividir a linha em seus componentes e verificar isoladamente um por um. Cada instrução pode ser colocada em uma linha adicional. O procedimento é simples e bastante útil, sobretudo quando a situação parece muito confusa e obscura.

Rode o programa mais uma vez para ver qual das novas linhas do grupo causa a mensagem de erro e retire-a do programa.

PARTE POR PARTE

Declarações individuais muito longas apresentam um tipo diferente de problema e, talvez, a melhor maneira de uni-las seja determinar o valor real de cada parte delas.

Mais uma vez, considere a hipótese de ter cometido um erro banal, antes de buscar interpretações complicadas. Se possível, reescreva a linha defeituosa do programa, de modo que obtenha um número de linhas separadas que possam ser tratadas individualmente. Esta é uma boa razão para se deixar espaços entre os números de linhas.

Onde você não puder fazer isto, "exploda" mentalmente a instrução e pergunte-se o que cada entrada está fazendo naquele determinado ponto do programa. Tente calcular quais os valores que foram obtidos para todas as variáveis em uso neste exato ponto.

Com o erro de sintaxe, os valores de qualquer variável em ação são irrelevantes — eles não poderão ser a causa do problema. Assim, mude a linha à vontade, mesmo que isso limpe as variáveis.

Se estiver lidando com um problema muito obscuro, o valor (ou valores) real da variável poderá fornecer os indícios. Utilize o computador para imprimir esses indícios no modo direto (imediato), antes de efetuar qualquer modificação no programa.

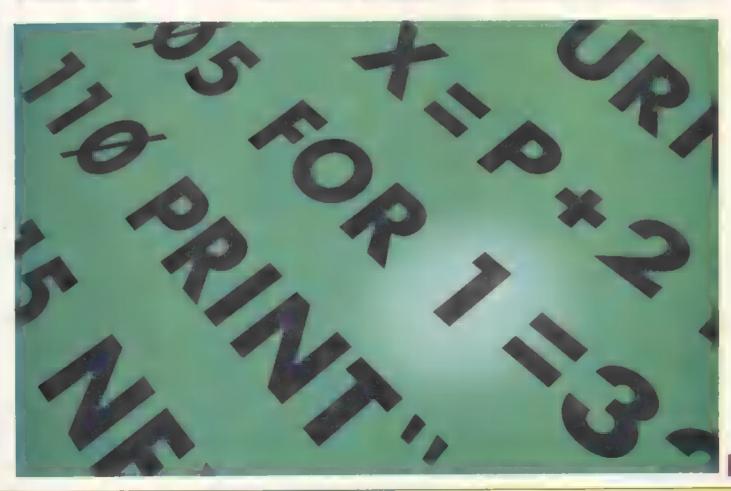
Simultaneamente, verifique o nome da variável, para se certificar de que está correto. Suponhamos que você habitualmente utilize a variável I para um loop de atraso de tempo. É bem provável que a introduza no programa sem querer, quando estiver copiando uma listagem e se deparar com um loop parecido, mas que utiliza uma variável diferente — um T, por exemplo.

ORIENTAÇÕES D[illegible]

Muitas mensagens de erro não fornecem indicações claras do número da linha errada. Em vez disso, simplesmente indicam onde um erro teve algum efeito. Quando isso ocorre, pode ser difícil apontar a causa com precisão.

Mas existem exceções. Como no exemplo acima, as mensagens **DATA** de erro são exibidas em uma linha que contém uma instrução incluindo **READ** quando, na verdade, o próprio erro se encontra na linha de instrução **DATA**.

Outras mensagens são menos evidentes (veja a tabela). No entanto, a maioria delas refere-se a erros que ocorrem antes do número das linhas apostrofadas na execução do programa. Essa é uma distinção importante, pois o erro pode estar, de fato, em uma sub-rotina situada antes ou depois da linha indicada na declaração de erro. Assim, uma variável poderá apanhar um valor incorreto bem antes de identificá-lo.

A melhor maneira de lidar com erros menos óbvios é examinar a ação do programa. Inicie imprimindo o valor de cada variável. Se você possuir uma impressora conectada ao computador, poderá

efetuar uma cópia deles. Mas é fácil utilizar o comando direto **PRINT** ou qualquer abreviação adequada, seguida pelo nome da variável — tal como **PRINT V** — e anotar os valores.

Examine, depois, como as variáveis funcionam em relação uma à outra. Novamente, poderá ser bastante útil dividir uma linha muito longa do programa em linhas mais curtas, cada qual contendo uma única instrução.

Pesquise cada variável a partir desse ponto do programa, para comparar seu valor real (o número que você anotou) com o que poderia ter se o programa rodasse corretamente.

Em um programa muito extenso, o trabalho será difícil e demorado. Parte dele poderá ser evitada se você rodar o programa em vários estágios distintos — imediatamente antes e depois de um determinado conjunto de entradas, por exemplo. Observe, então, as modificações dos valores das variáveis.

A não ser que você esteja familiarizado com o uso das teclas <**RUN/STOP**> ou <**BREAK**> do seu computador, é aconselhável introduzir no programa linhas provisórias contendo a declaração **STOP**. Mas note que, com esse procedimento, você limpará a memória e, assim, deverá rodar o programa mais uma vez — o que nem sempre será possível.

Quando o programa parar, imprima os valores das variáveis e, em seguida, tecle a instrução para continuar até o próximo **STOP**.

ERROS COMUNS

Entre as causas mais comuns de erros de programa estão as falhas cometidas no código de entrada, a partir de listagens impressas.

O problema específico é gerado sobretudo pela confusão entre caracteres parecidos: casos típicos são as letras l em tipo minúsculo, I em tipo maiúsculo e o número 1, assim como a letra O e o número 0.

É fácil também confundir, sobretudo em listagens de pouca qualidade, dois pontos com ponto e vírgula, e ponto final com vírgula. O ponto e vírgula é utilizado para a formatação da exibição, e o uso acidental de dois pontos (sinal que indica o fim de uma declaração), como no exemplo abaixo, poderá resultar simplesmente em uma mensagem de "ERRO DE SINTAXE":

```
95 INPUT "ENTRE SEU NOME":N$
```

A confusão entre o ponto final e a vírgula, porém, é ainda mais difícil de se identificar. Veja estas duas linhas:

```
1000 DATA 12.4,6,7,8,13,1.7
1000 DATA 12.4,6,7,8.13,1.7
```

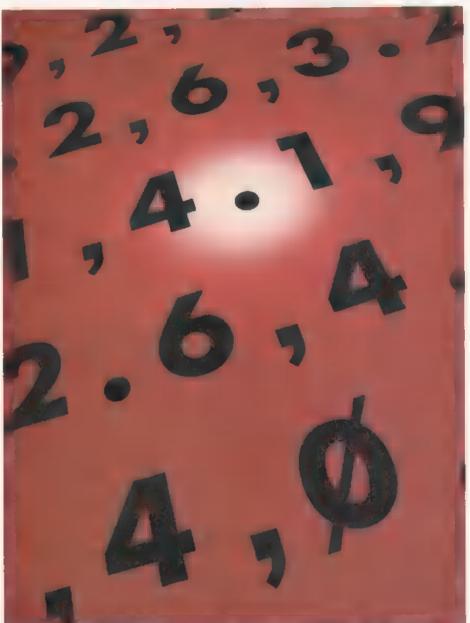
Tanto uma quanto outra pareceriam corretas, até mesmo depois de um minucioso exame. Note que existem seis itens de dados na primeira linha, mas apenas cinco na segunda.

Este poderia ser um indício de erro, caso as outras declarações **DATA** contivessem um número idêntico de itens. Incidentalmente, portanto, teríamos um método simples de realizar pelo menos uma verificação.

Se uma mensagem de erro exibiu "FIM DE DADOS", não existem itens de dados suficientes; portanto, a segunda linha poderia estar incorreta.

A presença de muitos itens, por sua vez, não provoca uma mensagem de erro; simplesmente assinala os valores incorretos para a variável **READ** ou, ainda, o valor incorreto para algumas das variáveis, se uma outra linha de dados for curta.

Uma mensagem de erro pode também resultar em uma linha **READ** ou uma linha de cálculo, se o tipo incorreto do valor for assinalado para uma variável **READ**.

O uso da letra O em vez do número 0 — ou vice-versa — pode acarretar um efeito idêntico.

ESPAÇOS

Deve-se deixar espaços em determinados lugares do programa, mas eliminá-los totalmente em outros — o problema, quase sempre, é reconhecer que algo tão "transparente" seja ■ causa de um problema.

Os espaços utilizados por razões estéticas — para separar, por exemplo, uma declaração impressa de outra ou tornar ■ listagens um pouco mais claras — não ocasionam erros se forem omitidos. Mas suspeite deles se a exibição na tela parecer achatada ou tiver partes superpostas.

Uma mensagem de erro ocorre se, acidentalmente, você incluir um espaço entre certas palavras-chave ■ o argumento que as segue colocado entre parênteses. Por exemplo: **TAB(n)** está correto; **TAB (n)**, não. As condições que determinam erros relacionados ■ espaço variam de um computador para outro, mas vale a pena verificar esse aspecto se não detectar falhas de outro tipo.

ERROS ESPECÍFICOS

Cada computador apresenta um tipo peculiar de erro, quase sempre relacionado ■ imperfeições — e não erros, propriamente — no sistema de operações ou no próprio hardware. Com freqüência, a falha se encontra no próprio BASIC e certas palavras-chave não se comportam adequadamente sob determinadas circunstâncias. Algumas das mais comuns estão detalhadas aqui, e serão apontadas em INPUT sempre que possam causar problemas inesperados, caso não se tome o devido cuidado.

FAÇA UM TESTE

Eis aqui um teste para ■ sua habilidade em detectar erros e uma oportunidade para colocar em prática tudo o que aprendeu. Os programas que ■ seguem estão cheios de erros — do tipo que poderia ser introduzido durante a cópia de uma listagem ou na adaptação de um programa sobre cuja ação não se tem um conhecimento completo. Eles oscilam de erros simples de digitação ■ sintaxe a erros na estrutura do próprio programa.

A versão correta do programa foi publicada na página 195 de INPUT. Adicionamos aqui uma linha extra para fazer com que o programa seja rodado novamente. Mas não olhe o programa original até que tenha elaborado o seu próprio caminho para localizar os erros.

O programa é um exemplo bem curto de como se deve distribuir os objetos entre ■ compartimentos de um jogo de aventuras. A lista dos objetos é lida ■ partir de uma lista de dados organizados em um conjunto; os números dos compartimentos estão armazenados em outro conjunto. A parte principal do programa — as linhas 70 ■ 100 — designa, aleatoriamente, ■ objeto para cada compartimento, verifica se todos foram utilizados e se há apenas um em cada compartimento. A linha 110 simplesmente imprime ■ resultado. Pelo menos é isto o que o programa deveria fazer se estivesse correto.

É provável que você identifique vários erros imediatamente, por meio da simples leitura do programa. Tente consertar todos os que puder e, então, quando estiver com ■ programa limpo, digite-o em seu computador e experimente rodá-lo. Certamente restarão alguns erros escondidos que você não identificou na primeira vez.

Se não conseguir resolver tudo, volte à página 195; de qualquer maneira, a versão correta da última linha deverá ser elaborada por você. E tenha o cuidado de não introduzir novos erros quando estiver corrigindo os outros!

```
10 LET g=14
20 DIM a$(g,7): DIM a(g)
30 FOR z=1 TO g
40 READ a$(g)
50 LET a(z)=z
70 FOR x=g TO 1 STEP -I
80 LET q=INT (RND*x)+1
90 LET t=a(x): LET a(x)=a(q):
 LET a(q)=T
100 NEXT x
110 FOR t=1 TO g: PRINT "A sal
a;t;tem";a$(a(t)): NEXT t
120 ■■■■ corda,"espada","espan
ador","faca","revolver","chave
""tocha","carro","lanterna","m
achado","bomba","livro","aerom
odelo","robo"
130 GOTO 10
```

```
10  LET G = 14
20  DIM A$(G),A(G)
30  FOR Z = 1TOG
40  READ A$(G)
50  LET A(Z) = Z
70  FOR ■ = G TO 1 STEP  - ■
80  LET ■ =  INT ( ■■■ (1) * X)
 + 1
90  LET T = A(X): LET A(X) = A(
Q): LET A(Q) = T
100  NEXT X
110  FOR T = 1 TO G: PRINT "A S
ALA";T"TEM";A$(A)(T)): NEXT T
120  DATA   CORDA,ESPADA,ESPANA
DOR,FACA,ARMA,CHAVE,TOCHA,CARRO
,LIQUIDIFICADOR,SOFA,BOMBA,LIVR
O,AEROMODELO,ROBO
130  GOTO 10
```

Para o MSX, adicione a linha abaixo:

```
5 R =  RND ( - TIME)
```

```
10 LET G=14
20 DIM A$(G),A(G)
30 FOR Z=1 TO G
40 READ A$(G)
50 A(Z)=Z
70 FOR X=G TO 1 STEP-I
■■ Q=RND(X)
90 T=A(X):A(X)=A(Q);A(Q)=t
100 NEXT X
110 FOR T=1 TO G:PRINT"NA SALA"
;T"HA"A$(A)(T)):NEXT T
120 DATE CORDA,ESPADA,ESPANADOR
,FACA,ARMA,CHAVE,TOCHA,CARRO,LA
NTERNA,MACHADO,BOMBA,LIVRO,AERO
MODELO,ROBO
```

FIGURAS MÓVEIS

O ZX-81 não tem muitos recursos gráficos. O Apple, embora os tenha com boa qualidade e resolução, define as figuras móveis por meio de um processo bastante complicado.

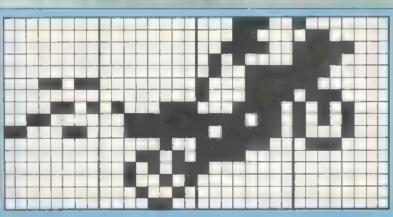
Com o código de máquina podemos obter alguns efeitos visuais no ZX-81. Um programa editor facilitará a criação de figuras móveis no Apple.

Enquanto outros micros usam uma codificação bem simples para criar figuras em alta resolução — números binários, onde "1" e "0" correspondem a pontos "acesos" e "apagados" —, o Apple tem um código complexo, que não explicaremos agora.

Neste artigo, os usuários do Apple ■ do ZX-81 verão como produzir figuras móveis. Usando os programas e observando as instruções, poderão, em seguida, criar ■■■ próprios desenhos.

- EDITOR DE FIGURAS
- MOTO E SUBMARINO NO APPLE
- COMO MOVIMENTAR OS DESENHOS
- UM MONSTRO NO ZX-81

Para facilitar o trabalho, o programa a seguir cria uma "tabela de figuras" na memória do micro, ■ partir de padrões desenhados com asteriscos dentro de linhas **DATA**.

```
100  ■■■■ :E = 35000: HIMEM: E:
 DIM ■(100),B(100)
110  HTAB 10: VTAB 12: PRINT "U
M INSTANTE, POR FAVOR"
120  FOR I = 35000 TO 37000: PO
KE I,0: NEXT : ■■■■
130 F =  INT (E / 256)
140  POKE 232,E - F * 256: POKE
 233,F
150  PRINT "(DEFINICAO DA TABEL
A) LINHA DATA ";: PRINT "20 ,0
";: FOR II = 0 TO 19:W =  INT (
(42 + II * 32) / 256):O = 42 +
II * 32 - ■ * 256: PRINT ",";O;
" ,";W;" ";: NEXT II: PRINT : P
RINT : PRINT "RETURN P/ CONTINU
AR": INPUT "";O$: HOME
160  POKE E,20: POKE E + 1,0: F
OR II = ■ TO 19
170 W =  INT ((42 + II * 32) /
256):O = 42 + II * 32 - W * 256
: POKE E + 2 + ■ * II,O: POKE E
 + 3 + 2 * II,W
180  FOR ■ = 0 TO 100:B(I) = 0:
 NEXT ■
190 Q = 0: ■■ : COLOR= 1: FOR I
 = ■ TO 8: READ Z$: FOR J = 1 T
O 8
200  IF  MID$ (Z$,J,1) = "*" TH
EN  PLOT J + 15,I + 29
210  NEXT J: NEXT I
220  HTAB 17: VTAB 21: PRINT "1
2345678"
230  GOTO 500
240  FOR V = 0 TO K - 1
250  IF B = 2 AND A(V) > 0 AND
A(V) < 4 THEN 280
260  IF B < 2 AND (A(V) > 0 OR
A(V) > 4) THEN 280
270 B = 0:Q = Q + 1
280 B(Q) = B(Q) + A(V) * (8 ^ B
)
290 B = B + 1
300  IF B > 2 THEN ■ = 0:Q = ■
+ 1
310  NEXT V
320  TEXT : HOME : PRINT "(FIGU
RA ";II + 1;") LINHA DATA  ";
330  FOR V = ■ TO 31
340  IF V <  > 0 THEN  PRINT ",
";
350  PRINT B(V);" ";
360  POKE E + 42 + 32 * II + V,
B(V)
370  NEXT V
380  PRINT : PRINT : PRINT "RET
URN P/ CONTINUAR": INPUT "";O$
390  HOME : HGR : SCALE= 1: ROT
= 0
400  FOR I = 150 TO 75 STEP  -
5
410  HCOLOR= 3
420  DRAW II + 1 AT 130,I
430  HCOLOR= 0
440  DRAW II + 1 AT 130,I
450  NEXT I
460  HCOLOR= ■
470  DRAW II + 1 AT 130,75
480  FOR I = 1 TO 50: NEXT I
490  NEXT II: END
500 X = 16:Y = 30:K = 1: FOR J
= 1 TO 4
510  FOR I = 1 TO 8
520 M = 1: IF  SCRN( X,Y) = 1 T
HEN M = M + 4
530 X = X + 1:A(K) = M:K = K +
1
540  NEXT I
550 M = 2: IF  SCRN( X,Y) = 1 T
HEN M = M + 4
560 Y = Y + 1:A(K) = M:K = K +
1
570  FOR I = 1 TO ■
580 M = 3: IF  SCRN( X,Y) = 1 T
HEN M = M + 4
590 X = X - 1:A(K) = M:K = K +
1
600  NEXT I
610 M = 2: IF  SCRN( X,Y) = 1 T
HEN M = M + 4
620 Y = Y + 1:A(K) = M:K = K +
1
630  NEXT J
640  GOTO 240
1000  REM    12345678
1001  DATA  "        "
1002  DATA  "        "
1003  DATA  "        "
1004  DATA  "        "
1005  DATA  "        "
1006  DATA  "        "
1007  DATA  "   **   "
1008  DATA  "      **"
```

O programa permite que você crie figuras móveis, tomando como modelo os desenhos que aqui apresentamos. Esses desenhos são destinados a outros micros, onde as figuras compõem-se de blocos de oito por oito pontos.

Para utilizar o programa, adicione ao seu final linhas **DATA** correspondentes às figuras da moto das páginas 316 e 317. Cada linha **DATA** deve conter o padrão de uma das linhas de um bloco de oito por oito pontos, desenhado com asteriscos. Na listagem encontram-se as oito primeiras linhas **DATA**, correspondentes ao primeiro bloco da moto. Digite os blocos na seguinte ordem: bloco superior esquerdo, bloco inferior esquerdo, bloco superior da segunda coluna, e assim por diante, até o bloco inferior direito. Podem ser digitados até vinte blocos, ou 160 linhas **DATA**.

Para montar na memória do Apple uma tabela de figuras, digite **RUN**. O mesmo comando imprimirá as linhas **DATA** necessárias para criar a tabela de dentro de um programa **BASIC**. Os desenhos dos blocos digitados serão executados em baixa e alta resolução.

COMO UTILIZAR A TABELA DE FIGURAS

Tendo a tabela com os blocos dos desenhos da moto na memória — são dezesseis blocos ou 128 linhas **DATA** —, você poderá dar movimento a eles. Digite **NEW** e copie o programa:

```
10   HGR : SCALE= 1: ROT= 0
100   FOR I = 0 TO 110 [illegible] 16
110   HCOLOR= [illegible]
120   DRAW 1 AT I,100
130   DRAW 2 AT I,108
140   DRAW 3 AT I + 8,100
150   DRAW 4 AT I + 8,108
160   DRAW 5 AT I + 16,100
170   DRAW 6 AT I + 16,108
180   DRAW 7 AT I + 24,100
190   DRAW 8 AT I + 24,108
200   FOR J = 1 T[illegible] 200: NEXT J
210   HCOLOR= 0
220   DRAW 1 AT I,100
230   DRAW 2 AT I,108
240   DRAW 3 AT I + 8,100
250   DRAW 4 AT I + 8,108
260   DRAW 5 AT I + 16,100
270   DRAW 6 AT I + 16,108
280   DRAW 7 AT I + 24,100
290   DRAW 8 AT I + 24,108
300   NEXT I
310   FOR I = 110 TO 250 [illegible]TE[illegible] 16
320   HCOLOR= 3
330   [illegible] 9 AT I,100
340   DRAW 10 AT I,108
350   DRAW 11 AT I + 8,100
360   DRAW 12 AT I + 8,108
370   DRAW 13 AT I + 16,100
380   DRAW 14 AT I + 16,108
390   DRAW 15 AT I + 24,100
400   DRAW 16 AT I + 24,108
410   FOR J = 1 TO 100: NEXT J
420   HCOLOR= 0
430   DRAW 9 AT I,100
440   DRAW 10 AT I,108
450   DRAW 11 AT I + 8,100
460   DRAW 12 AT I + 8,108
470   DRAW 13 AT I + 16,100
480   DRAW 14 AT I + 16,108
490   DRAW 15 AT I + 24,100
500   DRAW 16 AT I + 24,108
510   NEXT I
```

Na linha 10 **HGR** liga a tela de alta resolução; **SCALE = 1** define a escala do desenho e **ROT = 0** define a orientação horizontal do mesmo.

As linhas 120 a 190 desenham a moto, usando **DRAW N AT X,Y**, onde N é o número do bloco de oito por oito pontos, na ordem em que foram criados pelo programa editor; **X** e **Y** são as coordenadas da posição desejada. A linha 200 promove uma pequena pausa, antes que as linhas 220 a 290 apaguem os mesmos blocos, desenhando-os em preto — **HCOLOR = 0**. O **FOR...NEXT** das linhas 100 a 300 repete o processo, dando movimento ao desenho. O restante do programa é uma repetição da primeira parte, só que utiliza o desenho da moto "empinando" — blocos 9 a 16.

Para montar o desenho sem empregar o programa editor, adicione ao programa as linhas **DATA** que ele criou. As linhas seguintes usarão os números ali contidos para montar a mesma tabela de figuras.

```
20   HOME :E = 35000: HIMEM: E
30 F =  INT (E / 256): POKE 232
,E - [illegible] * 256: POKE 233,F
40   FOR I = E TO E + 41 + 16 *
32
50   [illegible] A: POKE I,A
[illegible]   NEXT
```

E contém o endereço inicial do local onde a tabela será montada na memória — é igual à variável de mesmo nome no programa editor. **HIMEM:E** protege esta área no topo da memória. Os **POKE** da linha 20 informam o endereço ao Apple. O restante do programa lê os dados com **READ** e monta a tabela com **POKE**. Na linha 40, 16 é o número total de blocos.

Outra alternativa para evitar o uso do editor é gravar a tabela de figuras em fita, usando o comando **W** do monitor, ou em disco, usando o comando **BSAVE**. As linhas **DATA** deixarão de ser necessárias, mas continuaremos precisando das linhas 20 a 40.

SUBMARINO

O programa seguinte movimenta o submarino da página 319, desde que ele tenha sido criado pelo programa editor.

```
10   HGR : SCALE= 1: ROT= 0
100 Y = 100:LY = Y: GOTO 170
110   GET A$: IF A$ = "" THEN 11
0
120 LY = Y
130   IF [illegible] = "P" AND Y > 3 THEN
 Y = Y - 3: GOTO 170
140   IF A$ = "L" AND [illegible] < 150 TH
EN Y = Y + 3: GOTO 170
150   IF A$ = " " THEN 260
160   GOTO 110
170   HCOLOR= 0
180   DRAW 1 AT 10,LY
190   [illegible] AT 18,LY
200   DRAW 3 AT 26,LY
210   HCOLOR= 5
220   DRAW 1 AT 10,Y
230   DRAW 2 AT 18,Y
240   DRAW 3 AT 26,Y
250   GOTO 110
260   FOR I = 0 TO 230 STEP 4
270   HCOLOR= 1
280   [illegible] 4 AT 34 + I,Y
290   HCOLOR= [illegible]
300   [illegible] 4 AT 34 + I,Y
310   NEXT I: GOTO 110
```

As teclas R e L movimentam o sub-

marino para cima e para baixo. A barra de espaço lança torpedos.

O programa que se segue cria um monstro com auxílio do código de máquina.

```
 1 REM ....................
10 LET A$="2A0C40233ABC4047112
100FE00280319"
15 LET A$=A$+"10FD3ABD4016005F
1911BE403ABB40FE"
20 LET A$=A$+"00280311CE400604
C506041A77231310"
25 LET A$=A$+"FA011D0009C110F0
C9010000"
30 LET A$=A$+"0000000000000000
0000000000000000"
35 LET A$=A$+"878B880480850580
0280800106850586"
50 FOR N=16514 TO 16605
60 POKE N,16*CODE A$+CODE A$(2
)-476
70 LET A$=A$(3 TO )
80 NEXT N
```

A linha 1 precisa ter pelo menos 92 caracteres após **REM**, para acomodar o

programa em código que será criado. As linhas 10 a 25 contêm o programa que coloca ■ monstro na tela — os códigos estão dentro de A$. Os zeros — 32 ao todo — da linha 30 apagam o desenho com espaços em branco.

A linha 35 contém o padrão do monstro. Se compararmos seu conteúdo — dois dígitos de cada vez — com o conjunto de caracteres do apêndice A do manual, veremos que ali estão caracteres gráficos invertidos.

Esses caracteres desenham o monstro, do canto superior esquerdo ao inferior direito. Como eles dividem um espaço de caractere em quatro partes, a resolução final é de oito por oito pontos.

Para mudar o desenho, basta trocar os números nesta linha. Códigos de comandos BASIC que tenham mais de um caractere com AT, TAB, THEN ou LEN não podem ser empregados.

Quando utilizamos **RUN**, o programa contido em A$ é colocado no espaço que se segue a **REM** pelas linhas 50 ■ 80. Se listarmos, então, o programa, veremos os caracteres correspondentes aos códigos. Os que desenham o monstro estão no final da linha.

Depois que o programa estiver dentro da linha **REM**, apague as demais e digite:

```
10 LET X=14
20 LET Y=8
30 POKE 16571,1
40 POKE 16572,Y
50 POKE 16573,X
60 RAND USR 16514
100 LET A$=INKEY$
110 IF A$="" THEN GOTO 100
120 IF A$="Z" AND X>0 THEN LET X=X-1
130 IF A$="X" AND X<28 THEN LET X=X+1
140 IF A$="P" ■■■ Y>0 THEN LET Y=Y-1
150 IF A$="L" AND Y<20 THEN LET Y=Y+1
160 POKE 16571,0
170 RAND USR 16514
180 GOTO 30
```

As linhas 10 e 20 colocam ■■ coordenadas do centro da tela em **X** e **Y**. A linha 30 introduz ■ número 1 no programa em código, usando **POKE**, para que seja impresso ■ monstro, ■ não espaços em branco. As linhas 40 e 50 colocam **X** ■ **Y** da mesma forma, estabelecendo ■ posição da figura. A seguir, a rotina em código é chamada e o computador desenha ■ monstro.

As linhas 100 a 150 contêm a rotina típica que move algo na tela (veja página 31). **Z** movimenta ■ desenho para ■ esquerda, ■ para a direita, **P** para cima e **L** para baixo.

A linha 160 coloca 0 dentro do programa em código, para desenhar espaços em branco; a linha 170 chama o programa, apagando o desenho. A linha 180 volta ao início.

■■■■■■ DA TELA

Com o código de máquina, pode-se também inverter ■ tela no ZX-81. O programa a seguir faz isso, trocando ■ preto pelo branco ■ vice-versa. Embora ele deva ser chamado de dentro de um programa BASIC, coloque-o na memória usando ■ nosso monitor (veja página 92).

```
2A 0C 40 06 17 23 7E FE 76 28
05 C6 80 77 18 F5 10 F3 C9
```

Use **RAND USR "endereço inicial"** para rodá-lo. Como esse comando, sem um número ■■ frente, limpará ■ tela, o efeito não será interessante. Um programa simples dará melhores resultados.

```
20 LIST
30 RAND USR
```

Aqui, **RAND USR** deve ser seguido do endereço inicial do programa.

Podemos ainda colocar essa rotina de inversão dentro do programa do monstro. Os códigos serão adicionados a **A$**. Acrescente a linha:

```
40 LET A$=A$+"2A0C400617237EFE762805C6807718F510F3C9"
```

Note que não há espaço entre os códigos.

A linha 50 deve ser alterada para colocar os códigos extras dentro do **REM**.

```
50 ■■■ N=16514 TO 16624
```

A linha **REM** precisa ter mais 19 espaços disponíveis, pelo menos. Após rodar o programa com essas modificações, apague todas as linhas, com exceção da **REM**, ■ copie o programa de movimentação com esta linha ■ mais:

```
155 IF A$="I" THEN ■■■■ USR 16606
```

O endereço 16606 é ■ início da nova rotina. Ela poderá ser chamada por meio da tecla I. Ao pressioná-la, o monstro será invertido. Mantendo a pressão, ele cintilará.

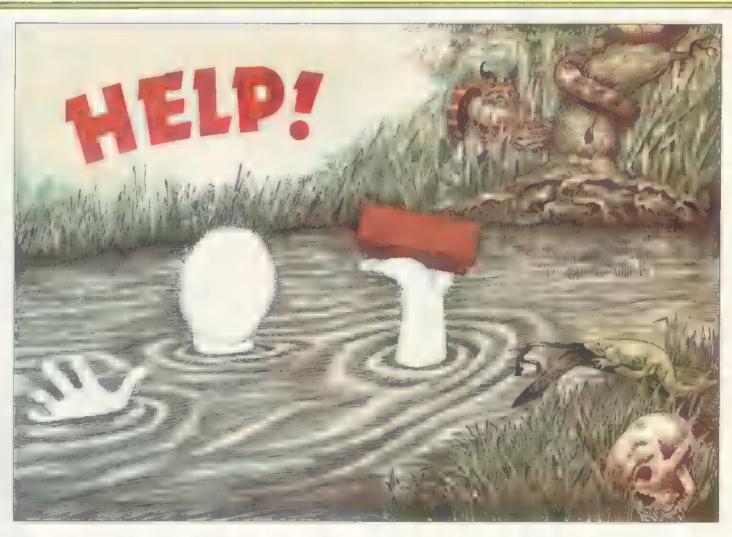
SIMULE ALTA RESOLUÇÃO

Como você já sabe, ■ ZX-81 não tem alta resolução gráfica. Porém, com o código de máquina, pode-se produzir algo semelhante. Este programa simplesmente ■■ **POKE** para colocar um pequeno programa após o **REM** da linha 10 e, depois, ■ chama.

```
10 REM ............
20 POKE 16514,62
30 POKE 16516,237
40 POKE 16517,71
50 POKE 16518,201
60 FOR N=0 TO 30
70 POKE 16515,N
80 ■■■■ USR 16514
90 NEXT ■
```

Infelizmente, não há um processo fácil para mover desenhos desse tipo.

A CONCLUSÃO DA AVENTURA

- COMO SOCORRER O HERÓI
- ELIMINE O COLETOR DE IMPOSTOS
- COMO ACENDER A LÂMPADA
- INSTRUÇÕES

Na busca incessante do olho perdido do totem inca, o herói chega à sala do trono de um reino remoto. Sairá ele com vida ou perecerá nas garras de um implacável coletor de impostos?

Depois de muitas peripécias e lances de emoção, aproxima-se do fim nosso jogo de aventura. Antes de concluí-lo, porém, devemos acrescentar-lhe certos detalhes, como novos perigos e avisos, além de uma saída pela qual o aventureiro possa escapar com vida. Algumas instruções também precisam ser incorporadas ao jogo.

Como essas rotinas contêm detalhes específicos a esta aventura, o programa que se segue não deve ser usado em outras aventuras sem modificações. Ele apenas completará o jogo em andamento e mostrará, em termos gerais, o que é necessário para fazer um programa desse tipo. No próximo artigo, mostraremos como os princípios apresentados aqui podem ser adaptados de modo a se encaixar às suas próprias idéias.

VOCÊ PRECISA DE AJUDA

Sempre que se encontrar diante de uma situação particularmente perigosa, o jogador (e, portanto, o personagem da aventura) pode recorrer ao computador, solicitando sugestões. Estas aparecerão na forma de mensagens impressas pela máquina em resposta ao pedido ("**AJUDAR**") do jogador. O significado de tais mensagens, assim como o momento em que elas serão impressas, depende apenas do programador. Podemos escolher, se desejarmos, não apresentar qualquer mensagem, ou fazê-las propositalmente ilusórias, ou mesmo prestar ajuda somente em alguns locais isolados.

Em nossa aventura existem vários pontos onde talvez valha a pena digitar

curtas mensagens para o jogador, advertindo-o, por exemplo, sobre o quarto escuro quando ele se encontrar em suas proximidades. Uma forma interessante de mensagem consiste em responder [illegible] pedido de "AJUDAR" com um "OLHE PARA ONDE VAI", ou algo até mais enigmático.

Outra mensagem pode ser emitida às margens do rio, onde o aventureiro corre o risco de se afogar (principalmente se estiver carregando um tijolo).

Evidentemente, o número dessas mensagens pode ser muito grande. Tudo depende da vontade do programador. Suponhamos, contudo, que este queira emitir apenas um aviso, por exemplo, no rio. O primeiro passo para isso consistirá em relacionar o aviso com o número do local — número 7 — e com a variável que registra a presença do tijolo, **OB (7)**.

```
3100 REM **AJUDAR**
3110 IF L<>7 OR B(2)<>-1 THEN
PRINT "DESCULPE, NAO POSSO AJUD
AR AGORA": GOTO 330
3120 PRINT "TIJOLOS SAO MUITO P
ESADOS. SEU  BRACO DEVE [illegible] D
OENDO.": GOTO 330
```

```
3100 REM **AJUDAR**
3110 IF L<>7 OR OB(2)<>-1 THEN
PRINT "DESCULPE, NAO POSSO AJUD
[illegible] AGORA":GOTO 330
3120 PRINT "TIJOLOS SAO MUITO P
ESADOS. SEU BRACO DEVE [illegible] DO
ENDO.":GOTO 330
```

Se o aventureiro não se encontrar no rio (**L < > 7**) ou não estiver carregando o tijolo — (**OB (2) < >−1** ou **B (2) < >−1**) —, a linha 3110 imprimirá [illegible] mensagem "DESCULPE, NÃO POSSO AJUDAR AQUI". Se, pelo contrário, ele estiver transportando o tijolo [illegible] solicitar ajuda ao chegar ao rio, então aparecerá o aviso "TIJOLOS SÃO MUITO PESADOS. SEU BRAÇO DEVE ESTAR DOENDO", impresso pela linha 3120.

Nada nos impede de acrescentar uma lista completa de mensagens de ajuda nas mais diversas condições e locais de aventura.

O COLETOR DE IMPOSTOS

Se o jogador é o "mocinho" da aventura, quem faz o papel de "bandido" é o coletor de impostos. O objetivo deste último é confiscar alguns objetos para saldar a dívida do nosso herói com o fisco. Assim, ele escolhe ao acaso um dos objetos transportados pelo jogador e o confisca, podendo levar até mesmo um tijolo como pagamento.

Se o aventureiro não estiver carregando alguma coisa quando o coletor de impostos aparecer, ele será preso e seu destino será apodrecer em uma masmorra. E assim o jogo terminará.

O papel do coletor de impostos é dar um toque de suspense ao jogo, já que ele surge de forma inesperada, independentemente do local ou de outras condições. Como em outros exemplos de acaso na programação de jogos, devemos fazer uso da função **RND**. O coletor é tratado como qualquer um dos objetos, com a diferença que sua localização é determinada ao acaso.

Aqui estão as linhas extras, para fazer o coletor de impostos aparecer.

```
320 IF  RND<(1/15)  AND  TA=0
THEN  LET B(7)=L: LET TA=1
480 IF B(7)=L [illegible] I<>10 THEN
GOTO 1590
```

```
320 IF RND(15)=1 AND TA=0 THEN
OB(7)=L:TA=1
480 IF OB(7)=L AND I<>10 THEN 1
590
```

```
320 RN=RND(-TIME):IF INT(RND(1)
*15+1)=1 [illegible] TA=0 [illegible] OB(7)=L:
TA=1
480 IF OB(7)=L [illegible] I<>10 THEN 1
590
```

```
320  IF  INT ( RND (1) * 15 + 1
) = 1 AND TA = 0 THEN OB(7) = L
:TA = 1
[illegible]  IF OB(7) = L AND I <  > 10
[illegible] 1590
```

Inicialmente, o Spectrum iguala todas essas variáveis a 0.

A linha 320 de todos os programas dá ao aventureiro uma chance em quinze de encontrar o coletor — este é o propósito do 15 no **RND**. O coletor só pode aparecer uma vez durante o jogo; portanto, você precisa da variável **TA** para saber se isso aconteceu ou não.

Se o número randômico for 1 (ou menor de que 1/15 no Spectrum) e [illegible] inspetor ainda não tiver aparecido, a linha

320 ajustará o valor do elemento 7 da matriz de localização de objetos — correspondente ao coletor —, de acordo com [illegible] lugar em que se encontra [illegible] jogador — **L**. O aviso de que o coletor chegou será dado como se ele fosse um objeto que tivesse sido colocado naquele local. Esse aviso nada mais é do que a descrição do objeto 7: o coletor.

A linha 480 tem [illegible] ver com o ato de eliminar o coletor, verificando se você tentou matá-lo. Se você não o fez, o programa pulará para a linha 1590.

COMO SE LIVRAR DO COLETOR

Quando o coletor de impostos mostrar sua cara feia, o aventureiro deverá

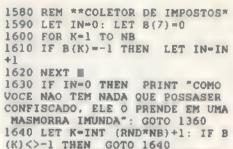
atirar nele com ■ revólver que pode ser encontrado ■■ outro lado do rio:

```
1540 REM **MATAR**
1550 IF B(4)<>-1 THEN  PRINT "C
OM O QUE?": GOTO 320
1560 IF B(7)<>L THEN  PRINT V$:
" QUEM?": GOTO 320
1570 PRINT "VOCE MATOU O ";B$(7
): LET B(7)=0: GOTO 330
```

```
1540 REM **MATAR**
1550 IF OB(4)<>-1 THEN PRINT"CO
M O QUE ?":GOTO 320
1560 IF OB(7)<>L THEN PRINT V$;
" QUEM ?":GOTO 320
1570 PRINT " VOCE MATOU O ";OB$
(7):OB(7)=0:GOTO 330
```

Essa rotina será usada quando o jogador digitar "MATAR" ou "ATIRAR". Se ele estiver sem ■ arma (**OB (4) < > −1**) ■■ (**B (4) < > −1**), ■ linha 1550 perguntará "COM QUÊ?". Se o coletor não estiver e o jogador tentar matá-lo, a linha 1560 imprimirá "QUEM?". A linha 1570 comunica: "VOCÊ MATOU O COLETOR" e ajusta ■ elemento 7 da matriz de localização de objetos para o inspetor não mais existir.

A VINGANÇA DO COLETOR

Aqui, o aventureiro é quem sofre nas mãos do coletor de impostos:

```
1580 REM **COLETOR DE IMPOSTOS*
1590 LET IN=0: LET B(7)=0
1600 FOR K=1 TO NB
1610 IF B(K)=-1 THEN  LET IN=IN
+1
1620 NEXT ■
1630 IF IN=0 THEN  PRINT "COMO
VOCE NAO TEM NADA QUE POSSASER
CONFISCADO, ELE O PRENDE EM UMA
 MASMORRA IMUNDA": GOTO 1360
1640 LET K=INT (RND*NB)+1: IF B
(K)<>-1 THEN  GOTO 1640
1650 PRINT "ELE TOMA ■ ";B$(K),
"DE VOCE": LET B(K)=0: GOTO 400
```

```
1580 REM **COLETOR ■■ IMPOSTOS*
1590 IN=0:OB(7)=0
1600 FOR K=1 TO ■■
1610 IF OB(K)=-1 THEN IN=IN+1
1620 NEXT
1630 IF IN=0 THEN PRINT "COMO V
OCE ■■■ TEM ■■■■ QUE POSSASER C
ONFISCADO, ELE O PRENDE  EMUMA
MASMORRA IMUNDA":GOTO 1360
1640 K=RND(NB):IF OB(K)<>-1 THE
■ 1640
1650 PRINT "ELE TOMA O ";OB$(K)
," DE VOCE":OB(K)=0:GOTO 330
```

```
1580  REM  ** COLETOR DE IMPOST
OS **
1590 ■■ = 0:OB(7) = 0
1600  FOR ■ = 1 TO NB
1610  IF OB(K) =  - 1 THEN IN =
 IN + 1
1620  NEXT
1630  IF ■■ = 0 THEN  PRINT "CO
MO VOCE ■■■ ■■■ ■■■■ QUE POSSA
SER    CONFISCADO, ELE O PRENDE
 EM UMA          MASMORRA IMUNDA
": GOTO 1360
1640 ■ =  INT ( RND (1) * NB +
1): IF OB(K) <  >  - 1 THEN 164
0
1650  PRINT "ELE TOMA O ";OB$(K
);" DE VOCE":OB(K) = 0: GOTO 33
0
```

Como ao inspetor de taxas é permitido aparecer apenas uma vez durante a aventura, a linha 1590 ajusta o elemento 7 da matriz de localização de objetos de forma que ele não mais exista enquanto durar o programa. Esse procedimento não afeta em nada a rotina, mas evita ■ aparecimento do coletor no futuro. IN é um marcador usado para verificar se objetos estão sendo carregados.

As linhas 1600 e 1620 passam pela matriz de localização de objetos, verificando se cada objeto está sendo transportado. Qualquer objeto que estiver

sendo carregado aumenta 1 no marcador IN.

Se nada estiver sendo transportado, então o valor de IN permanecerá 0 e o aventureiro será comunicado: "COMO VOCÊ NÃO TEM NADA QUE POSSA SER CONFISCADO, ELE O PRENDERÁ EM UMA MASMORRA IMUNDA". O jogo termina e pergunta-se ao aventureiro se ele deseja uma outra jogada; isto é feito pela linha 1360.

Caso haja objetos sendo transportados, a linha 1640 pegará um deles ao acaso. Se esse objeto estiver entre os que estão sendo carregados, ele será apreendido. No entanto, se não estiver, outro objeto será escolhido a esmo e assim por diante, até que a escolha coincida com um que esteja sendo transportado.

Se um objeto adequado tiver sido selecionado, a linha 1650 comunicará ao aventureiro que objeto foi confiscado pelo coletor. O elemento da matriz de localização de objetos correspondentes será então alterado; assim, o objeto não mais existirá.

UM POUCO DE NATAÇÃO

Esta rotina é usada quando o jogador decide atravessar o rio:

```
1400 REM **NADAR**
1410 IF L<>7 THEN  PRINT "NADAR
 ONDE ?!!": GOTO 400
1420 IF B(2)=-1 THEN  PRINT "QU
E VERGONHA, VOCE SE AFOGOU !":
GOTO 1360
1430 IF B(4)>-1 THEN  PRINT "VO
CE ACHOU UM REVOLVER": LET B(4)
=-1: GOTO 400
1440 PRINT "VOCE SE MOLHOU TODO
": GOTO 400
```

```
1400 REM **NADAR**
1410 IF L<>7 THEN PRINT " ONDE?
!!":GOTO 330
1420 IF OB(2)=-1 THEN PRINT " Q
UE VERGONHA, VOCE SE AFOGOU!":G
OTO 1360
1430 IF OB(4)>-1 THEN PRINT " V
OCE ACHOU UM REVOLVER":OB(4)=-1
:GOTO 330
1440 PRINT " VOCE SE MOLHOU TOD
O":GOTO 330
```

A linha 1410 verifica se o aventureiro está próximo ao rio. Se não estiver, o computador perguntará: "NADAR ONDE?!!" Como não existem piscinas ou praias na aventura, não há razão para escrever uma rotina de entrada para lidar com qualquer outra resposta. Nenhuma sugestão aparecerá e o jogo prosseguirá, mostrando as direções disponíveis. Caso tente atravessar o rio carregando o tijolo, o jogador morrerá — "QUE VERGONHA; VOCÊ SE AFOGOU!", comentará friamente o computador. Mas essa "morte" tem suas particularidades: depois de afundar, o jogador poderá ressuscitar, se escolher jogar novamente.

A linha 1430 verifica se o jogador carrega o revólver. Em caso negativo, o elemento da matriz de localização de objetos correspondente ao revólver será ajustado e surgirá a mensagem: "VOCÊ ENCONTROU UM REVÓLVER".

Se o aventureiro já encontrou a arma e estiver tentando atravessar o rio novamente, a linha 1440 dirá: "VOCÊ SE MOLHOU TODO".

FINALMENTE, O OLHO PERDIDO

O jogador só poderá recuperar o tão cobiçado olho perdido do totem inca se o saco de bolas de gude tiver sido encontrado. O passo a seguir será, portanto, esvaziar o saco para o olho aparecer. Aqui está a rotina.

```
1450 REM **ESVAZIAR**
1460 IF N$<>"SACO"( TO LEN N$)
THEN  PRINT "ISTO NAO PODE SER
ESVAZIADO": GOTO 400
1470 IF B(1)<>-1 THEN  LET G=1:
 GOTO 1270
1480 PRINT "AS BOLINHAS SE ESPA
LHAM PELO CHAO": LET B(5)=L: GO
TO 370
```

```
1450 REM **ESVAZIAR**
1460 IN=INSTR("SACO",N$):IF IN<
>1 THEN PRINT " ISTO NAO PODE S
ER ESVAZIADO":GOTO 330
1470 IF OB(1)<>-1 THEN G=1:GOTO
 1270
1480 PRINT " AS BOLINHAS SE ESP
ALHAM PELO    CHAO":OB(5)=L:GOT
O 370
```

```
1450  REM  ** ESVAZIAR **
1460 IN = 0: IF N$ =  LEFT$ ("S
ACO", LEN (N$)) THEN IN = 1
1465  IF IN <  > 1 THEN  PRINT
" ISTO NAO PODE SER ESVAZIADO":
 GOTO 330
1470  IF OB(1) <  >  - 1 THEN G
= 1: GOTO 1270
1480  PRINT " AS BOLINHAS SE ES
PALHAM PELO CHAO":OB(5) = L: GO
TO 370
```

A rotina é chamada quando o aventureiro aciona o comando "ESVAZIAR" alguma coisa. A linha 1460 verifica se essa coisa é o saco. Se não for (N$ < > "SACO"), aparecerá a mensagem "ISTO NÃO PODE SER ESVAZIADO". A linha 1470 verifica se o saco está sendo carregado (OB (1) < >-1 ou B (1) < >-1). Se não estiver, em vez de emitir outra mensagem, o programa pulará para a linha 1270, que exibirá a mensagem: "VOCÊ NÃO PODE LARGAR O QUE NÃO TEM". Como o jogador não quererá "LARGAR" nada, essa linha deve ser modificada, de forma a dar uma resposta adequada. Mude-a para:

```
1270 IF B(G) < > -1 THEN PRINT
"VOCE NAO PODE ";V$;" O QUE NAO
 TEM":GOTO 330
```

```
1270 IF OB(G)<>-1 THEN PRINT "V
OCE NAO PODE ";V$;" O QUE NAO T
EM":GOTO 330
```

A variável V$ imprimirá o verbo adequado — "LARGAR" ou "ESVAZIAR" —, conforme a situação.

Se o saco estiver sendo carregado, o programa irá para a linha 1480. A mensagem "AS BOLINHAS SE ESPALHAM PELO CHÃO" será mostrada e o elemento 5 da matriz de localização de objetos será acertado.

Não há necessidade de imprimir uma mensagem para informar o resultado porque, ao pular para a linha 370, a impressão usual de uma descrição longa poderá substituí-la. Aparecerá então no vídeo a descrição usada na matriz de descrições longas (ver linha 240).

ACENDA A LÂMPADA

A lâmpada precisa ser acesa para que o jogador veja as saídas do quarto escuro. Se o aventureiro não estiver carregando a lanterna, a escuridão não será vencida e ele ficará preso. Esta é a rotina para acender a lâmpada.

```
1490 REM **ACENDER**
1500 IF N$<>"LAMPADA"( TO LEN N
$) THEN  PRINT "NAO PODE SER FE
ITO": GOTO 400
1510 IF B(6)<>-1 THEN  LET G=6:
 GOTO 1270
1520 IF LA=1 THEN  PRINT "JA ES
TA ACESA": GOTO 400
1530 LET LA=1: LET DA=0: PRINT
"OK": GOTO 330
```

```
1490 REM **ACENDER**
1500 IN=INSTR("LAMPADA",N$):IF
IN<>1 THEN PRINT " NAO PODE SER
 FEITO":GOTO 330
1510 IF OB(6)<>-1 THEN G=6:GOTO
 1270
1520 IF LA=1 THEN PRINT" JA EST
A ACESA":GOTO 330
1530 LA=1:PRINT "OK":GOTO 330
```

```
1490 ■■■ ** ACENDER **
1500 IN = 0: IF N$ = LEFT$ ("L
AMPADA", LEN (N$)) THEN ■■ = 1
1505 IF IN < > 1 THEN PRINT
" NAO PODE SER FEITO": GOTO 330
1510 IF OB(6) < > - 1 THEN G
 = 6: GOTO 1270
1520 IF LA = 1 THEN PRINT "JA
 ESTA ACESA": GOTO 330
1530 LA = 1: PRINT "OK": GOTO 3
30
```

Para que essa rotina seja requisitada, é preciso que o jogador emita a instru-

ção "**ACENDER**". A linha 1500 (1505 no Apple) é bem parecida com a linha equivalente na rotina "**ESVAZIAR**", verificando se [illegible] aventureiro escreveu a palavra "**LÂMPADA**". Caso a lanterna não esteja sendo carregada, surgirá a mensagem "VOCÊ NÃO PODE ACENDER O QUE NÃO TEM". A linha 1520 verifica se o "indicador de lâmpada acesa" — LA — está "ligado", comunicando o fato ao aventureiro. O indicador de lâmpada acesa é fixado em 1 pela linha 1530, que também imprime um "OK".

A AVENTURA [illegible]GA AO FIM

Quando nosso herói entra em cena, encontra uma corrente pendurada, próxima ao trono.

O que deve ele fazer? Que tal puxar a corrente? Aqui está uma rotina que cuidará das conseqüências:

```
1300 REM **PUXAR**
1310 IF N$="CORRENTE"( TO LEN N
$) THEN  LET IN=1: IF IN=1 AND
L<>24 THEN  PRINT "NADA ACONTEC
E": GOTO 400
1320 IF IN<>1 THEN  PRINT "VOCE
 NAO PODE PUXAR ISTO !": GOTO 4
00
1330 IF B(5)<>-1 THEN  PRINT "V
OCE CAI DENTRO DO VASO E VAI EM
BORA COM [illegible] DESCARGA": GOTO 1360
1335 REM **FIM DA AVENTURA**
1340 PRINT "PARABENS ! VOCE COM
PLETOU [illegible] TAREFA."
1360 PRINT '"QUER JOGAR NOVAMEN
TE (S/N)?"
1370 LET A$=INKEY$: IF A$<>"S"
AND A$<>"N" THEN  GOTO 1370
1380 IF A$="S" THEN  RUN
1390 STOP
```

```
1300 REM **PUXAR**
1310 IN=INSTR("CORRENTE",N$):IF
 IN=1 AND L<>24 THEN PRINT " NA
DA ACONTECE":GOTO 330
1320 IF IN<>1 THEN PRINT " VOCE
 [illegible] PODE PUXAR ISSO!":GOTO 330
1330 IF OB(5)<>-1 THEN PRINT" V
OCE CAI DENTRO DO VASO E VAI
EMBORA COM A DESCARGA":GOTO 136
0
1340 REM **FIM DA AVENTURA**
1350 PRINT "PARABENS! VOCE COMP
LETOU A TAREFA"
1360 PRINT:PRINT" QUER JOGAR NO
VAMENTE (S/N)?"
1370 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 1370
1380 IF A$="S" THEN RUN
1390 END
```

```
1300  REM  [illegible] **
1310 IN = 0: IF N$ =  LEFT$ ("C
ORRENTE", LEN (N$)) THEN IN = 1
1315  IF [illegible] = 1 AND L <  > 24 T
HEN  PRINT "NADA ACONTECE": GOT
O 330
1320  IF IN <  > 1 THEN  PRINT
" VOCE NAO [illegible] PUXAR ISSO": GO
TO 330
1330  IF OB(5) =  - 1 THEN 1340
1335  PRINT " VOCE CAI DENTRO D
[illegible] VASO E VAI EMBORA COM A DESCA
RGA": GOTO 1360
1340  REM  [illegible] FIM [illegible] AVENTURA [illegible]
1350  PRINT "PARABENS ! VOCE CO
MPLETOU A TAREFA.:? "FIMDAAVENT
URA"
```

```
1360  PRINT : PRINT " QUER JOGA
R NOVAMENTE (S/N) ?"
1370  GET A$: IF A$ < > "S" AN
D A$ < > "N" THEN 1370
1380  IF A$ = "S" THEN  RUN
1390  END
```

A linha 1310 considera a possibilidade de o jogador ter trazido a corrente consigo antes mesmo de puxá-la. Neste caso, ela dirá ao aventureiro que "NADA ACONTECE".

Se o herói tentar puxar qualquer outro objeto, a linha 1320 lhe dirá: "VOCÊ NÃO PODE PUXAR ISSO".

Depois disso, acontece o inesperado. Se o jogador estiver na sala do trono, não tendo porém encontrado o olho, aparecerá a mensagem "VOCÊ CAI DENTRO DO VASO E VAI EMBORA COM A DESCARGA". E o jogo termina.

Se o aventureiro der um jeito de encontrar o olho e puxou a corrente no aposento real, nenhuma dessas linhas terá efeito e ele poderá suspirar aliviado ao receber a mensagem: "PARABÉNS! VOCÊ COMPLETOU A TAREFA. FIM DA AVENTURA".

Finalmente, nas linhas 1360 e 1380, há uma opção para jogar novamente. Esta, porém, só será útil se o aventureiro tiver sido enclausurado na masmorra ou seguido com a descarga cano adentro.

AS INSTRUÇÕES

Devemos incorporar, agora, algumas informações à rotina do jogo. Antes disso, porém, precisamos verificar o espaço restante da memória (ver página 212). Se esse espaço for pequeno, será necessário retirar todas as linhas REM — teremos então que remunerar os comandos **GOSUB** que direcionam o programa para essas linhas, evitando assim mensagens de erro.

A decisão de quantas instruções serão fornecidas deve ser tomada de acordo com o espaço disponível de memória. Nelas podemos incluir até considerações quanto ao formato do vídeo do computador, o que influenciará na quantidade de detalhes a serem inseridos antes de passarmos para outra tela.

Como nossa aventura é muito simples, a rotina de instruções reduz-se a algumas linhas, contendo poucas informações. Aqui está:

```
  80 CLS : PRINT "QUER INSTRUCO
ES (S/N)?"
  90 LET A$=INKEY$: IF A$="" TH
EN  GOTO 20
  95 IF A$="S" THEN  GOSUB 6000
6000 REM **INSTRUCOES**
6010 CLS : PRINT : PRINT "  DEV
IDO A UM COLAPSO FINANCEIRO  VO
CE DEIXOU O PAIS."
6020 PRINT : PRINT "  SEUS PROB
LEMAS VAO TERMINAR      QUANDO V
OCE  ENCONTRAR  O       LEGENDA
RIO OLHO CRAVEJADO DE     BRILHA
NTES DE UM TOTEM INCA.     DEPOI
S DE FAZE-LO,VOCE TERA     QUE
ENCONTRAR A SAIDA."
6030 PRINT : PRINT "    CUIDADO
 COM O COLETOR              DE
IMPOSTOS !"
6040 PRINT AT 20,4;"PRESSIONE Q
UALQUER TECLA         PARA CONTI
NUAR"
6050 LET A$=INKEY$: IF A$="" TH
EN  GOTO 6050
6060 RETURN
```

```
10 CLS:PRINT" QUER INSTRUCOES
S/N)?"
20 A$=INKEY$:IF A$="" THEN 20
30 IF A$="S" THEN GOSUB 6000
6000 REM **INSTRUCOES**
6010 CLS:PRINT:PRINT " DEVIDO A
 UM COLAPSO FINANCEIRO  VOCE DE
IXOU O PAIS."
6020 PRINT:PRINT " SEUS PROBLEM
AS VAO TERMINAR      QUANDO VOCE
 ENCONTRAR O           LEGENDARIO
 OLHO CRAVEJADO DE      BRILHANTE
S DE UM TOTEM INCA.    DEPOIS D
E FAZE-LO VOCE TEM QUE  ENCONTR
AR A SAIDA."
6030 PRINT:PRINT"   CUIDADO  CO
M  O  COLETOR              DE
IMPOSTOS !"
6040 PRINT @451,"APERTE QUALQUE
R TECLA PARA      CONTINUAR."
6050 A$=INKEY$:IF A$="" THEN 60
50
6060 RETURN
```

```
10  HOME : PRINT "QUER INSTRUCO
ES (S/N)?"
20  GET A$: IF A$ = "" THEN 20
30  IF A$ = "S" THEN  GOSUB 600
0
6000  REM  ** INSTRUCOES **
6010  HOME : PRINT "DEVIDO A UM
 COLAPSO FINANCEIRO, VOCE "
6020  PRINT "DEIXOU O PAIS."
6025  PRINT "SEUS PROBLEMAS VAO
 TERMINAR"
6030  PRINT "QUANDO VOCE ENCONT
RAR O LEGENDARIO"
6040  PRINT "OLHO CRAVEJADO DE
BRILHANTES, DE UM "
6050  PRINT "TOTEM INCA. DEPOIS
 DE FAZE-LO, VOCE "
6060  PRINT "TERA QUE ENCONTRAR
 A SAIDA"
6070  PRINT : PRINT "CUIDADO CO
M O COLETOR DE IMPOSTOS !"
6080  PRINT : PRINT "APERTE QUA
LQUER TECLA PARA CONTINUAR"
6090  GET A$: IF A$ = "" THEN 6
090
6100  RETURN
```

Agora, grave em fita ou disco a aventura completa. A estrutura desse jogo — *O Olho Perdido do Totem Inca* — pode servir de base para suas próprias aventuras.

DATILOGRAFE FRASES LONGAS

Agora que você já tem certo domínio do teclado, chegou ■ momento de testar ■■ habilidade como datilógrafo, copiando frases mais longas exibidas na tela do computador.

Se você já se sente em condições de digitar qualquer palavra ou frase a um ritmo seguro, podemos passar a uma nova etapa de nosso curso de datilografia. Até este ponto, o curso foi útil tanto para aqueles que querem apenas digitar programas quanto para os interessados em escrever cartas ou usar um editor de textos.

Nesta seção, você encontrará ■ oportunidade de aperfeiçoar suas habilidades, utilizando um texto mais longo. Dessa forma, além de tornar-se mais rápido, você acumulará experiência para usar o computador como uma ferramenta de escrita.

É muito importante continuar fiel à técnica de digitação usada até agora: portanto, não olhe para ■ teclado enquanto digita; use sempre as teclas base como referência; procure também manter ■ ritmo constante — não saia digitando sofregamente, ■ alta velocidade, ■■ sem segurança (a regra de ouro é iniciar a uma velocidade que possa ser mantida por algum tempo e, só então, ir acelerando aos poucos).

COMO USAR O PROGRAMA

Ao executar o programa, ■ computador apresentará um menu inicial com duas opções de teste. A primeira mostra frases geradas ao acaso a partir de declarações **DATA** do programa. A segunda requer um pouco mais de trabalho. Ela permite que você use frases mais longas, mas exige que estas sejam digitadas antes. Feito isto, as frases são mostradas na tela, como no primeiro teste. Em ambos ■■ casos você deverá datilografar o texto da maneira que ele aparece na tela; o programa lhe informará então o número de erros cometidos e a sua velocidade em palavras por minuto. É possível também escolher a maneira de proceder ao se cometer um erro (isto é, você pode usar a tecla de retrocesso para corrigi-lo ou não).

Se for escolhida a primeira opção, o engano deve ser corrigido assim que ocorre.

De outro modo, o computador aguardará até que ■ tecla correta seja pressionada.

O programa para o Apple é um pouco diferente do de outros micros: ele não apresenta a velocidade de digitação devido à falta de um cronômetro interno. E se o seu Apple (ou compatível) não gera caracteres minúsculos, não há problema em digitá-los todos em caracteres maiúsculos.

Ao digitar as linhas que contêm as instruções **DATA**, tome cuidado em deixar sempre um espaço no fim de cada frase, para que as palavras não fiquem justapostas quando o programa for executado.

Após algum tempo usando o programa, você verá que ■■ frases se tornam conhecidas e monótonas, visto que há poucas variações. Tente então substituí-las por outras a seu gosto. Para fazer isso, liste as linhas **DATA** e reescreva-as com frases suas. Lembre-se de colocar entre aspas as que contêm vírgulas; faça o mesmo com as que terminam cada uma das linhas. Você evitará, assim, que elas sejam divididas em duas, ou que o espaço final seja desprezado pelo computador. Tome medidas também para que o número de frases não se altere. Se você não se sentir à vontade para fazer isto, pode variar o exercício optando por dar entrada a trechos completos de um texto, ou seja, optando pelo teste 2.

Cada passagem pode ter um máximo de 255 caracteres e ■ computador aceita até três frases ■■ mesmo tempo. Responda às perguntas que aparecerem e digite ■■ frases de sua escolha. Depois de colocá-las na memória do computador, você poderá escolher qualquer uma delas.

Nos micros MSX, TRS-Color e Spectrum, os erros são comunicados pelo programa por intermédio de um som grave. Nos computadores da linha Apple, essa função é desempenhada por um bip.

```
10 POKE 23561,0
20 CLS
25 DIM t$(3,255): DIM t(3):
LET df=0
30 PRINT AT 7,7;"Qual teste (
1 ou 2) ?"
```

- APRENDA A DIGITAR SENTENÇAS MAIS LONGAS
- VEJA COMO O PROGRAMA FUNCIONA
- USE A TECLA DE RETROCESSO PARA CORRIGIR OS ERROS
- PRATIQUE COM TEXTOS MAIS LONGOS
- REGRAS PARA UMA BOA APRESENTAÇÃO DO TEXTO

```
  40 PRINT AT 10,7;"Digite '0'
  para sair"
  50 LET a$=INKEY$: IF a$<"0"
  ■ a$>"2" THEN  GOTO 50
  60 IF a$="0" THEN  STOP
  70 GOSUB VAL a$*1000
  80 CLS : PRINT AT 15,4;"Palav
  ■■ por minuto= ";LEN c$*(INT
  ((500/(PEEK 23672+256*PEEK
  23673))*100)/100)
  90 PRINT AT 17,6;"Numero de ■
  rros= ";e
 100 GOTO 30
1000 CLS : PRINT "Deseja habili
tar ■ tecla DELETE (s/n)?"
1010 LET a$=INKEY$: IF a$<>"n"
■■■ a$<>"s" THEN  GOTO 1010
1020 LET e=0: LET d=0: IF a$="s
" THEN  LET d=1
1030 CLS
1040 LET c$="": RESTORE : FOR k
=1 TO 4: LET r=INT (RND*3)+1: ■
■■ j=1 TO 3
1050 ■■■■ b$: IF j=r THEN  LET
c$=c$+b$
1060 NEXT j: NEXT k
1070 PRINT c$: PRINT : PRINT
1080 LET pp=0
1090 LET a$=INKEY$: IF a$="" ■■
■■  GOTO 1090
1100 POKE 23672,0: POKE 23673,0
: PAUSE 0: GOTO 1120
1110 PAUSE 0
1115 LET a$=INKEY$: IF a$="" TH
■■  GOTO 1110
1120 IF a$<>c$(pp+1) AND d=0 TH
■■  GOTO 1170
1130 PRINT a$;CHR$ 95;CHR$ 8;:
LET pp=pp+1
1140 IF a$<>c$(pp) THEN  GOTO 1
170
1150 SOUND .01,30: IF pp=LEN c$
 THEN  RETURN
1160 GOTO 1110
1170 SOUND .05,-10: LET e=e+1
1180 IF d=0 THEN  GOTO 1110
1190 PAUSE 0: LET a$=INKEY$: IF
 a$<>CHR$ 12 THEN  GOTO 1190
1200 LET pp=pp-1: PRINT CHR$ 8;
CHR$ 95;" ";CHR$ 8;CHR$ 8;: GOT
O 1110
1210 GOTO 1130
1500 DATA "O cachorro manco que
 anda com tres pernas ","E fato
 que qualquer ■■ ","Na hora cer
ta, ■ elefante "
1510 DATA "pode arrastar ","ser
a capaz de sentar sobre ","pode
 pular por sobre "
1520 DATA "a velha caixa esbura
cada ","a torre inclinada de Pi
sa ","qualquer ■■■ das arvores
```

```
(1)="" AND A$(2)="" THEN 2090
2010 PRINT" VOCE QUER USAR UMA
PASSAGEM JA  DIGITADA (S/N) ?"
2020 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 2020
2030 IF A$="S" THEN 2120
2040 IF A$(P)="" THEN 2090
2050 P=P+1:IF P<3 THEN 2040
2060 PRINT " AS TRES PASSAGENS
FORAM DIGITA- DAS. QUAL VOCE DE
SEJA  REESCRE- VER (1-3)?"
2070 A$=INKEY$:IF A$<"1" OR A$>
"3" THEN 2070
2080 P=VAL(A$)-1 PRINT A$:PRINT
2090 POKE 282,0:PRINT "DIGITE U
MA PASSAGEM: "
2100 LINE INPUT A$(P)
2110 GOTO 2150
2120 CLS:PRINT"QUAL PASSAGEM SE
[illegible] USADA (1-3) ?"
2130 A$=INKEY$:IF A$<"1" OR A$>
"3" THEN 2130
2140 P=VAL(A$)-1:IF A$(P)="" TH
EN 2120
2150 CLS
2160 POKE 282,255:CLS:PRINT " D
ESEJA HABILITAR A TECLA DE RE
TROCESSO (S/N)?"
2170 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 2170
2180 D=0:E=0:IF A$="S" THEN D=1
2190 CLS:P[illegible] 282,0:C$=A$(P):GO
SUB 1070:RETURN
```

```
10  HOME : DIM A$(2)
20  HTAB 8: [illegible] 5: PRINT "QUAL
 TESTE? (1-2)"
30  HTAB 6: VTAB 7: PRINT "TECL
E <0> PARA TERMINAR.";
40  GET A$: IF A$ < "0" OR A$ >
 "2" THEN 40
50  IF [illegible] VAL (A$) THEN [illegible]
E : END
60  ON  VAL (A$) GOSUB 1000,200
[illegible]
70  [illegible] : HTAB 7: VTAB 14: PRI
NT "NUMERO DE ERROS =>";E
80  GOTO 20
1000  HOME :Y = 0
1010  PRINT "VOCE [illegible] [illegible] [illegible] T
ECLA                    DE RETRO
CESSO (<-)?";
1020  GET A$: IF A$ <  > "S" [illegible]
D A$ <  > "N" THEN 1020
1030 E = 0:D = 0: IF  ASC (A$)
= 83 THEN [illegible] = 1
1040  HOME : IF Y [illegible] 1080
1050 C$ = "": RESTORE : FOR K =
 1 TO 4:R =  INT ( RND (1) * 3)
 + 1: FOR J = 1 TO 3
1060  [illegible] B$: IF J = R THEN C$
 = C$ + B$
1070  [illegible] : NEXT
1080  PRINT C$:PP = 0: VTAB 12
1090  GET A$: IF [illegible] =  CHR$ (8)
[illegible] 1090
1100  IF A$ <  >  MID$ (C$,PP +
 1,1) [illegible] D = 0 THEN 1140
1110  PRINT A$;:PP = [illegible] + 1
1120  IF A$ <  >  MID$ (C$,PP,1
) [illegible] 1140
1130  IF PP =  LEN (C$) THEN [illegible]
ETURN
1135  GOTO 1090
1140  [illegible]  CHR$ (7);:E = E +
1
1150  IF  NOT D THEN 1090
1160  GET A$: IF A$ <  >  CHR$
(8) [illegible] 1100
1170  PRINT A$;:PP = PP - 1: GO
TO 1090
1500  [illegible]  O cachorro manco [illegible]
e anda com tres pernas ,E fato
que qualquer [illegible] ,"Na hora certa
, [illegible] elefante "
1510  DATA  pode arrastar ,sera
 capaz de sentar sobre ,"pode p
ular por sobre "
1520  DATA  a velha caixa esbur
acada ,a torre inclinada de Pis
a ,"qualquer uma das arvores da
 fazenda "
```

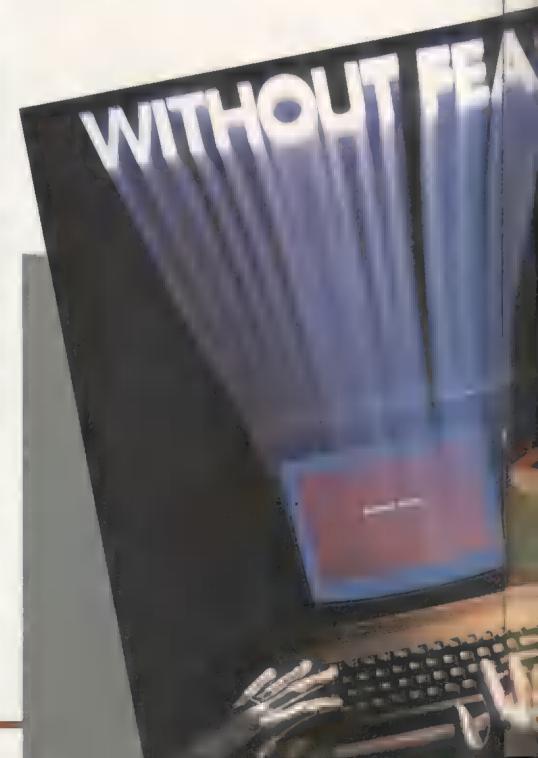
A APRESENTAÇÃO DE UM TEXTO

Dada [illegible] facilidade com que [illegible] pode corrigir [illegible] no teclado do computador, [illegible] possível produzir cartas limpas, sem borrões ou letras esbranquiçadas pelo uso de papel corretor.

Um documento bem montado apresenta margens de pelo menos três centímetros de ambos os lados (o texto, obviamente, deve estar centralizado [illegible] sentido vertical). Se você usar um editor de textos, [illegible] margens serão definidas automaticamente; mas, num programa seu, isso deve [illegible] feito por [illegible]termédio dos comandos **PRINT** ou **PRINT AT**.

Tão importante quanto as margens é [illegible] distribuição correta de parágrafos. Estes fazem com que o texto fique claro [illegible] fácil de [illegible] lido. Existem duas for[illegible] de iniciar um parágrafo: [illegible] tradicional, [illegible] que [illegible] primeira palavra começa [illegible] ser escrita alguns espaços para a direita; e a [illegible] moderna, na qual [illegible] pula uma linha entre um parágrafo [illegible] [illegible]tro, sem fazer [illegible] recuo no começo da frase. Pode-se também, evidentemente, combinar os dois métodos.

Se você tiver uma carta muito pequena para redigir, deixe [illegible] bom espaço na parte de cima do papel, antes de iniciar a impressão e/ou aumente [illegible] número de linhas entre os parágrafos. Com um pouco de prática, você produzirá textos bem centralizados e com um belo aspecto.

```
1530  DATA  e derruba-la com ■■
orme estrondo.,sem medo de ter
■■■ surpresa.,ate a hora de fec
har o zoologico.
2000  HOME :P = 0: IF A$(0) + ■
$(1) + A$(2) = "" THEN 2090
2010  PRINT "VOCE QUER USAR ■■■
 PASSAGEM JA DIGITADA? (S/N)";
2020  GET A$: IF ■■ <  > "S" ■■
D A$ <  > "N" ■■■■ 2020
2030  IF A$ = "S" THEN 2150
2040  IF A$(P) = "" ■■■■ 2090
2050 P = P + 1: IF P < 3 THEN 2
040
2060  PRINT : PRINT "AS TRES PA
SSAGENS JA ■■■■ DIGITADAS.
   QUAL VOCE DESEJA REESCREVER?
 (1-3)";
2070  GET A$: IF A$ < "1" OR A$
 > "3" THEN 2070
2080 P =  VAL (A$) - 1: PRINT A
$: PRINT
2085 A$(P) = ""
2090  PRINT : PRINT "DIGITE ■■■
 PASSAGEM: "
2100  FOR I = 1 TO 255
2110  GET A$: IF A$ =  CHR$ (13
) THEN 2140
2115  IF A$ =  CHR$ (8) THEN A$
(P) =  LEFT$ (A$(P), LEN (A$(P)
) - 1): PRINT  CHR$ (8);: GOTO
2110
2120 A$(P) = A$(P) + A$
2125  HTAB 1: VTAB 5: PRINT A$(
P);
2130  NEXT
2140  GOTO 2180
2150  HOME : PRINT "QUAL PASSAG
EM SERA USADA? (1-3) ";
2160  GET A$: IF A$ < "1" OR A$
 > "3" THEN 2160
2170 ■ =  VAL (A$) - 1: IF A$(P
) = "" THEN 2150
2180  HOME
```

```
2190 C$ = A$(P):Y = 1: GOSUB 10
10: RETURN
```

```
10 CLEAR2000
20 R=RND(-TIME):S$="L10 O2 G"
30 CLS:DIMA$(2)
40 LOCATE 8,5:PRINT"Qual teste?
 (1 ou 2)"
50 LOCATE 6,7:PRINT"Tecle <0> p
ara terminar."
60 A$=INKEY$:IFA$<"0"ORA$>"2"TH
EN60
70 IFA$="0"THENCLS:END
80 ONVAL(A$)GOSUB1000,2000
90 CLS:LOCATE5,12:PRINT"Palavra
■ por minuto => ";USING "###.##
";LEN(C$)*600/TIME
100 LOCATE7,14:PRINT"Número de
erros =>";E
110 GOTO40
1000 CLS:COLOR 15,6:Y=0
1010 PRINT"Você quer ■■■ ■ tec
la de             retrocesso
 (<<)?"
1020 A$=INKEY$:IFA$<>"s"ANDA$<>
"S"ANDA$<>"n"ANDA$<>"N"THEN1020
1030 E=0:D=0:IFA$="s"ORA$="S"TH
END=1
1040 CLS:IFYTHEN1080
1050 C$="":RESTORE:FORK=1TO4:R=
INT(RND(1)*3)+1:FORJ=1TO3
1060 READB$:IFJ=RTHENC$=C$+B$
1070 NEXT:NEXT
1080 PRINTC$
1090 PP=0
1100 A$=INKEY$:IFA$=""THEN1100
1110 TIME=0:GOTO1140
1120 A$=INKEY$:IFA$=""ORA$=CHR$
(8)THEN1120
1130 IFA$<>MID$(C$,PP+1,1)ANDD=
0THEN1170
1140 VPOKE PP+201,ASC(A$):PP=PP
+1
1150 IFA$<>MID$(C$,PP,1)THEN117
0
1160 BEEP:IFPP=LEN(C$)THENCOLOR
 15,4:RETURNELSE1120
1170 PLAY S$:E=E+1
1180 IFD=0THEN1120
1190 A$=INKEY$:IFA$=""THEN1190
1200 IFA$<>CHR$(8)THEN1130
1210 PP=PP-1:VPOKEPP+201,32:GOT
O1120
1500 DATA O cachorro ■■■■ que
anda com três pernas ,E fato qu
■ qualquer um ,"Na hora certa,
o elefante "
1510 DATApode arrastar ,será ■■
paz de sentar sobre ,"pode pula
r por sobre "
1520 DATAa velha caixa esburaca
da ,a torre inclinada de Pisa ,
"qualquer uma das árvores da fa
zenda "
1530 DATAe derrubá-la com enorm
e estrondo.,sem medo de ter ■■■
 surpresa.,até ■ hora de fechar
 ■ zoológico.
2000 CLS:P=0:IFA$(0)+A$(1)+A$(2
)=""THEN2090
2010 PRINT"Você quer usar uma ■
assagem já digitada? (S/N)"
2020 A$=INKEY$:IFA$<>"S"ANDA$<>
"s"ANDA$<>"n"ANDA$<>"N"THEN2020
2030 IFA$="S"ORA$="s"THEN2120
2040 IFA$(P)=""THEN2090
2050 P=P+1:IFP<3THEN2040
2060 PRINT:PRINT"As três passag
■■■ foram digitadas.        Qual
 você deseja reescrever? (1-3)"
2070 A$=INKEY$:IFA$<"1"ORA$>"3"
THEN2070
2080 P=VAL(A$)-1:PRINTA$:PRINT
2090 PRINT"Digite ■■■ passagem:
"
2100 LINEINPUTA$(P)
2110 GOTO2150
2120 CLS:PRINT"Qual passagem ■■
rá usada? (1-3)"
2130 A$=INKEY$:IFA$<"1"ORA$>"3"
THEN2130
2140 P=VAL(A$)-1:IFA$(P)=""THEN
2120
2150 CLS
2160 COLOR 15,12:C$=A$(P):Y=1:G
OSUB1010:RETURN
```

Deve-se seguir alguma regra ■■ modificar as frases do programa?

Isso depende de sua capacidade como datilógrafo. Se o curso inteiro foi bem assimilado, você deve ser capaz de digitar corretamente qualquer texto, seja este longo ou curto, simples ou complexo, usando os dez dedos ■ não apenas dois, como fazem alguns iniciantes.

Os dedos menores, no entanto, não raro exigem mais treinamento que ■■ outros. Para isso, certifique-se de que ■■■ frases contêm muitos Qs, As, Zs, Ps, e Ls. É possível fazer treinamentos extra de quaisquer dedos, montando um conjunto apropriado de palavras.

É preciso, contudo, que essas palavras apareçam misturadas a outras, de modo a cobrir todo o teclado. A frase "A zebra quase fugiu do zoológico pulando a grade" ■ um exemplo, assim como "Fique quieto ■ traga-me ■■■ caixa com doze litros de uísque". Se você quiser aumentar a destreza dos dedos menores, faça novos exercícios, inventando outras frases como essas.

O programa desta lição combina três frases para formar um texto que tenha algum sentido. Com um pouco de imaginação, você poderá conseguir o mesmo efeito com suas próprias frases.

BÚSSOLAS E RELÓGIOS

Seno, cosseno, tangente: transformadas em comandos do BASIC, funções trigonométricas são essenciais quando quer desenhar curvas, círculos e elipses no computador.

Os computadores são freqüentemente relacionados com Matemática. Embora programar em BASIC não exija muito conhecimento nessa área, a maioria dos processos do BASIC é familiar a qualquer matemático. Contudo, muitos deles são usados não só para fazer cálculos, como também para controlar várias outras operações.

Algumas das funções matemáticas mais úteis para programador são aquelas que calculam a relação entre ângulos e distâncias; elas são, aliás, as mais indicadas para o trabalho com gráficos.

Suponhamos, por exemplo, que queremos desenhar um relógio. Poderíamos começar pela caixa, usando o comando **CIRCLE** — menos no TRS-80 e no Apple —, mas teríamos dificuldade em posicionar corretamente os números, se calculássemos cada posição manualmente. Ora, um relógio com números em posições erradas não serviria para nada.

Felizmente, as funções matemáticas podem ser usadas para fazer esse cálculo. No relógio, a distância entre dois números é de um doze avos de volta. Essa distância pode ser calculada pelo computador: basta fornecer-lhe um número em graus ou em radianos.

Uma volta completa tem, no círculo, 360 graus, o que equivale a 2xPI radianos. No relógio, os números estão posicionados a cada 30 graus (30°), ou a cada PI/6 radianos — ou seja, cada um doze avos de volta. O número PI (lê-se "pi") é às vezes representado pela letra grega π.

Freqüentemente usado para calcular vários aspectos de um círculo, tais como sua área e comprimento, PI vale aproximadamente 22/7 (quase 3,14). Alguns computadores já têm esse número armazenado na memória (como os micros da linha Sinclair).

É aconselhável familiarizar-se tanto com graus como com radianos, pois, se os primeiros são a medida mais comum para ângulos, é com radianos que os computadores trabalham.

Se calcularmos **SIN 30** (seno de 30) ao mesmo tempo numa calculadora e num computador, os resultados serão expressos de maneira diferente, a não ser que a calculadora esteja no modo **RAD** (radiano). Isto acontece porque, enquanto esta trabalha com medidas em graus, o computador opera com radianos.

DE GRAUS PARA RADIANOS

Para passar um número de graus para radianos basta dividi-lo por 180 e depois multiplicar o resultado por PI. Para efetuar a operação inversa (de radianos para graus), faz-se o contrário: multiplica-se o número por 180 e divide-se por PI.

O programa a seguir o ajudará a familiarizar-se com as conversões. Ele converte graus em radianos e vice-versa. Se você tiver uma calculadora, e quiser comparar os resultados, use-a da seguinte maneira: calcule o seno (comando **SIN**), o cosseno (**COS**) a tangente (**TAN**) de um número na calculadora e anote os resultados; converta esse número em radianos no computador; faça os mesmos cálculos na calculadora e compare os resultados com os obtidos anteriormente — se tudo for feito corretamente, eles deverão ser iguais.

```
10 INPUT "DESEJA CONVERTER GR
AUS PARA      RADIANOS (1) OU
RADIANOS       GRAUS (2) ?";a
20 IF a-2 THEN  GOTO 70
30 IF a<>1 THEN  GOTO 10
40 INPUT "QUAL E O NUMERO?";b
50        "E IGUAL A "; b/180*
```

- CONVERTA GRAUS EM RADIANOS
- COMO DESENHAR UMA BÚSSOLA
- COMO MEDIR ÂNGULOS NUMA BÚSSOLA
- O QUE SIGNIFICAM SIN, COS E TAN
- OS GRÁFICOS DE [illegible] E COS
- USE SIN E COS PARA DESENHAR CÍRCULOS
- UMA ESFERA FEITA DE ELIPSES

A divisão do círculo em graus possibilita uma grande variedade de aplicações, como os minutos e horas de um relógio ou os pontos cardeais de uma bússola. Uma volta completa contém 360 graus. No computador, essa contagem é feita por meio de radianos: cada volta abriga, neste caso, 2 PI radianos (PI é a forma fonética [illegible] letra grega π, que designa [illegible] constante matemática de valor aproximado [illegible] 3,14).

```
PI;" RADIANOS": GOTO 90
70 INPUT "QUAL E [illegible] NUMERO?";b
80 PRINT "E IGUAL [illegible] "; b*180/
PI;" GRAUS"
90 PRINT "PRESSIONE QUALQUER
TECLA PARA   COMECAR NOVAMENT
E": PAUSE 0: CLS : GOTO 10
```

TT

```
10 PI=4*ATN(1)
20 CLS
30 INPUT"VOCE QUER CONVERTER [illegible]
AUS [illegible] [illegible] (1)  OU RADI
[illegible] PARA  GRAUS (2) ";A
40 IF A=2 THEN GOTO 90
50 IF A<>1 THEN GOTO 20
60 [illegible] "QUAL [illegible] O [illegible] ";B
70 [illegible] "ISSO E IGUAL [illegible] ";B*1
/180;" RADIANOS"
[illegible] GOTO 110
90 INPUT "QUAL [illegible] O NUMERO ";B
100 PRINT "ISSO E IGUAL A ";B*1
80/PI;" GRAUS"
110 PRINT "PRESSIONE QUALQUER T
ECLA PARA   EXECUTAR OUTRA VEZ.
"
120 A$=INKEY$:IF A$="" THEN GOT
O 120
130 GOTO 20
```

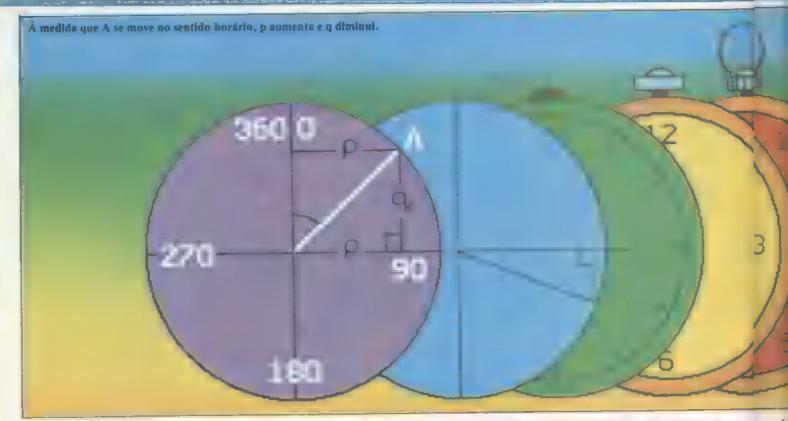
```
10 PI=4*ATN(1)
20 CLS
30 INPUT"CONVERSÃO [illegible] [illegible] PAR
A RADIANOS(1) OU [illegible] RADIANOS PA
RA GRAUS(2) ";A
40 IF A<1 OR A>2 THEN 10
50 PRINT:INPUT"QUAL [illegible] NUMERO ";
B
60 PRINT:PRINT"ELE VALE";
70 IFA=1THEN PRINT B/180*PI;"RA
DIANOS"
80 IFA=2THEN PRINT B*180/PI;"GR
AUS"
[illegible] PRINT:PRINT"QUALQUER TECLA P
[illegible] REPETIR"
100 IF INKEY$="" [illegible] 100
110 GOTO 20
```

```
10 PI = [illegible] (1)
20  HOME
30  [illegible] "CONVERSAO DE GRAUS P
[illegible] RADIANOS(1) OU  DE RADIANOS
 PARA GRAUS(2) ? ";A
40  IF A <  > 1 AND A <  > 2 TH
EN 20
50  PRINT : INPUT "QUAL O [illegible]
O ? ";B
60  PRINT : PRINT "ELE VALE ";
70  IF A = 1 [illegible]  PRINT [illegible] / 18
0 * PI;" RADIANOS"
80  IF A = 2 THEN  PRINT B * [illegible]
0 / PI;" GRAUS"
90  PRINT : PRINT "QUALQUER TEC
LA PARA REPETIR"
100  [illegible] A$: IF A$ = "" THEN 10
0
110  GOTO 20
```

Uma vez rodado, o programa pede por 1 ou 2, que especificam se a conversão é de graus em radianos ou de radianos em graus. Depois o programa solicita [illegible] número a ser convertido [illegible] faz aparecer [illegible] resultado na tela. Entretanto, visualizar um ângulo que é representado por um simples número pode às vezes ser muito difícil, a não ser que estejamos em condições de vê-lo desenhado. Uma representação bastante comum de todos [illegible] ângulos possíveis é encontrada nas bússolas. Poderíamos verificar is[illegible] numa bússola de verdade; mas para quê, se o computador pode mostrá-lo também?

DESENHE UMA BÚSSOLA

Como existe uma forte relação entre círculos (ou arcos de círculos) e medidas de ângulo, podemos usar essas medidas para desenhar relógios, bússolas

À medida que A se move no sentido horário, p aumenta e q diminui.

ou quaisquer outras figuras circulares.

Os programas mencionados a seguir desenham uma bússola e posicionam as marcações de graus. O norte corresponde a 0, o leste a 90, ■ sul a 180 e o oeste ■ 270 graus. Construída a bússola, o programa desenhará qualquer ângulo que quisermos.

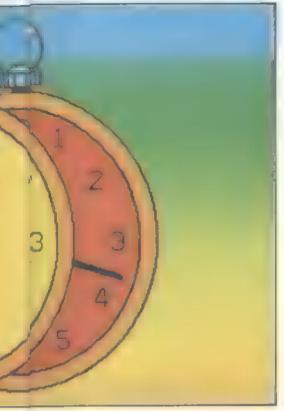
Somente a versão para o Sinclair Spectrum imprime números em volta da bússola. Mais adiante veremos como contornar ■ problema.

Quando rodamos o programa, ■ computador pede por um número e traça uma linha do centro do círculo até um certo ponto, formando um ângulo de valor equivalente ao número fornecido. Isso quer dizer que, se fornecermos 90, a linha desejada partirá do centro em direção à direita. Se digitarmos 180, a linha sairá do centro e irá para baixo, representando assim um ângulo de 180 graus.

Note que o computador espera que forneçamos um número em graus para depois convertê-lo em radianos. Se olharmos para cada programa, veremos que a variável de entrada é dividida por 180 e multiplicada por PI. Isto nos poupa o trabalho de fazer a conversão manualmente.

```
10 BORDER 4: PAPER 4: INK 0:
CLS
20 CIRCLE 131,88,60
30 PLOT 131,84 : ■ 0,8:
 PLOT 127,88: DRAW 8,0
40 FOR a=0 TO 2*PI STEP PI/4
50 PLOT 131+55 ■ SIN a,88+55*
COS a: DRAW 10*SIN a,10*COS a
60 NEXT ■
70 PRINT AT 2,16;0
80 FOR b=45 TO 360 STEP 45
90 PRINT AT 10-10*COS (b/180*
PI),15+10*SIN (b/180*PI);b
100 NEXT b
110 INPUT "   QUE ANGULO, EM G
RAUS, VOCE        DESEJA VER?",c
120 INK 2
130 PLOT 131,88: DRAW 45*SIN (
c/180*PI),45*COS (c/180*PI)
140 INK 0
150 INPUT " QUER RECOMECAR (s/
n)?";d$
160 IF d$="s" THEN  PLOT 131,
88: DRAW  OVER 1;45*SIN (c/180
*PI),45*COS (c/180*PI)
170 IF d$<>"s" THEN  BORDER 7:
PAPER 7: CLS : STOP
180 PLOT 131,84: DRAW 0,8:
PLOT 127,88: DRAW 8,0: GOTO
110
```

```
10 PMODE 4,1
20 PCLS
30 PI=4*ATN(1)
40 CIRCLE(127,95),80,5
50 FOR X=0 TO 2*PI STEP PI/4
60 LINE(127+72*SIN(X),95-72*COS
(X))-(127+79*SIN(X),95-79*COS(X
)),PSET
70 NEXT ■
80 CLS: INPUT"QUE ANGULO VOCE Q
■■■ ";Z
90 SCREEN 1,1
100 X=127+60*SIN(Z*PI/180):Y=95
-60*COS(Z*PI/180)
110 LINE (127,91)-(127,99),PSET
120 LINE (123,95)-(131,95),PSET
130 LINE (127,95)-(X,Y),PSET
140 IF INKEY$="" THEN 140
150 LINE(127,95)-(X,Y),PRESET
160 GOTO ■■
```

```
10 PI=4*ATN(1)
20 CLS
30 INPUT"QUE ANGULO DESEJA VER
";Z : Z=Z/180*PI
40 SCREEN2:COLOR1,15
50 CIRCLE(128,96),80,1,,,1
60 FOR X=0 TO 2*PI STEP PI/4
70 LINE(128+72*SIN(X),96-72*COS
(X))-(128+79*SIN(X),96-79*COS(X
)),1
80 ■■■ X
90 LINE(128,92)-(128,100),1
100 LINE(124,96)-(132,96),1
110 FOR I=0 TO 500:NEXT
120 X=128+60*SIN(Z):Y=96-60*COS
(Z)
130 LINE(128,96)-(X,Y),1
140 FOR I=0 TO 3000:NEXT
150 GOTO 30
```

```
10  HGR
20  HCOLOR= 3: HOME
30 PI = 4 [illegible]  ATN (1)
40  FOR [illegible] = 0 TO [illegible] [illegible] PI STEP PI
 / 60
50  HPLOT 140 + 80 *  SIN (X),8
0 - 80 *  COS (X)
60  NEXT X
70  FOR [illegible] = [illegible] TO [illegible] * PI STEP PI
 / 4
80  HPLOT 140 + 72 *  SIN (X),8
[illegible] - 72 *  [illegible] (X) TO 140 + 79 *
  [illegible] (X),80 - 79 [illegible]  COS (X)
90  NEXT [illegible]
100  HPLOT 140,76 TO 140,84
110  HPLOT 136,80 TO 144,80
120  VTAB 23
130  INPUT "QUE ANGULO DESEJA [illegible]
[illegible] ? ";Z:Z = Z / 180 [illegible] PI
140 X = 140 + 60 [illegible]  [illegible] (Z):Y =
 80 - 60 *  [illegible] (Z)
150  HPLOT 140,80 TO X,Y
160  GET A$: IF A$ = "" THEN 16
[illegible]
170  HCOLOR= 0
180  HPLOT 140,80 TO X,Y
190  HOME : HCOLOR= [illegible]
200  GOTO 100
```

Os programas do Apple e do TRS-Color começam ajustando o modo gráfico adequado na linha 10; depois, eles limpam a tela. Como o TRS-Color, o MSX e o Apple não têm o número PI armazenado na memória; seus programas criam uma variável, PI, com o mesmo valor do número PI.

Todos os programas, exceto o do Apple, usam depois o comando **CIRCLE** para desenhar a caixa da bússola (círculo) — linha 40 no TRS-Color, linha 20 no Spectrum e linha 50 [illegible] MSX. O Apple não possui o comando **CIRCLE**; por isso, nele o círculo é desenhado passo a passo nas linhas 40 [illegible] 60.

Os programas desenham então as marcações que completam a bússola: pequenas linhas a cada 45 graus ao redor do círculo nos ajudam a saber a exata posição de cada marcação (a versão pa[illegible] o Spectrum numera as posições). Tudo isso é posicionado de acordo com [illegible] abertura de cada ângulo.

A parte seguinte dos programas, menos o do MSX, pede que entremos o ângulo desejado: linha 80 para o TRS-Color, 110 para o Spectrum [illegible] 130 para o Apple e o TK-2000.

Uma pequena cruz no centro do círculo desenhado nos ajudará [illegible] verificar se os ângulos estão corretos: se entrarmos um ângulo de 90 graus, [illegible] traço lateral direito da cruz será superposto pela nova linha. A cruz é desenhada por meio de duas pequenas linhas.

Mostrado o ângulo pedido, a versão do Spectrum pergunta se queremos ver outro ângulo; a versão dos outros micros espera que apertemos qualquer tecla. Antes de desenhar um novo ângulo, o TRS-Color, [illegible] Spectrum e o Apple apagam primeiramente a linha anterior. O MSX apaga todo o desenho e o refaz depois com o ângulo novo.

Para ampliar [illegible] analogia com [illegible] bússola basta substituir as marcações de graus por N, L, S e O (ou seja, norte, leste, sul e oeste), e usar o programa como um indicador de direção. Assim, se quisermos viajar num ângulo de 270 graus, devemos entrar o valor 270; o computador desenhará então uma linha, apontando a direção desejada.

PONTOS NUM CÍRCULO

Os programas apresentados na seção anterior usam **SIN** e **COS**, duas funções BASIC que correspondem às funções trigonométricas "seno" e "cosseno", respectivamente.

Elas [illegible] referem [illegible] posição de um ponto na circunferência em relação a dois eixos que se cortam perpendicularmente [illegible] centro do círculo. O eixo vertical é chamado de "Y" e o horizontal, de "X". Na ilustração destas páginas, [illegible] ponto A da circunferência está ligado aos dois eixos pelas linhas p e q.

Neste caso, p [illegible] q têm o mesmo comprimento. Mas, se movermos A para baixo, sobre [illegible] circunferência, perceberemos que p cresce enquanto q diminui. Quando A chegar aos 90 graus, q valerá O, enquanto p será igual ao raio da circunferência.

Usando o programa da bússola podemos visualizar melhor o que acontece. Entremos os ângulos 0, 30, 45 e 90. À medida que as linhas mudam de posição, podemos imaginar como p e q, embora não desenhadas na tela, variam de comprimento. Com uma régua poderíamos medir p e q direto da tela.

O VALOR DE SIN E COS

Evidentemente, [illegible] relação entre p e q muda à medida que o ângulo aumenta ou diminui. Existe ainda uma relação entre o raio do círculo e as linhas p e q (quando A estiver em 90 graus, p terá o mesmo valor do raio do círculo). Assim como a anterior, essa relação entre o raio e as linhas p e q também pode ser calculada, mudando sempre que [illegible] ângulo mudar.

A relação entre o raio e p é chamada de "seno" do ângulo. A relação entre o raio e q é [illegible] "cosseno" do ângulo. Ou seja, se dividirmos as linhas p e q pelo raio, obteremos, respectivamente, [illegible] seno e o cosseno desse ângulo. Assim, se o raio for igual a 1, os valores do seno e do cosseno serão iguais aos comprimentos de p e de q.

O triângulo da ilustração abaixo foi obtido do diagrama da bússola. Um dos seus vértices é formado pelo encontro do eixo X com a linha que liga A ao centro do círculo. Uma terceira linha, que sai de A e encontra o eixo X em ângulo reto, forma [illegible] último lado do triângulo.

O seno do ângulo formado pelo encontro da linha que liga A ao centro do círculo com o eixo X é [illegible] relação entre o lado *oposto* a esse ângulo [illegible] a hipotenusa. A hipotenusa é sempre o lado oposto ao ângulo reto. Na ilustração ela é formada pela linha branca que liga A ao centro do círculo. O terceiro lado do triângulo é chamado de *adjacente*.

O cosseno também é, como vimos, uma relação entre dois lados. Existe ainda uma outra relação entre lados, conhecida como tangente do ângulo.

Em resumo, as relações são as seguintes:

seno = lado oposto/hipotenusa
cosseno = lado adjacente/hipotenusa
tangente = lado oposto/lado adjacente

As três relações podem ser calculadas pelo computador por intermédio das funções **SIN**, **COS** [illegible] **TAN**. Por exemplo, **PRINT SIN.5** coloca na tela o se-

no do ângulo .5 radianos.

Na ilustração da página 336, à medida que o ponto se move em sentido horário, partindo do topo do círculo (ângulo 0), os valores do seno e do cosseno aumentam e diminuem respectivamente, porque p e q mudam de tamanho. Depois dos 90 graus o seno começará [illegible] diminuir; quando o ponto passar pelos 180 graus tudo mudará outra vez.

Lembremos que o seno mede o quanto um ponto está *à direita* do eixo Y. Assim, quando ele atingir a metade esquerda do círculo, o seno [illegible] tornará *negativo*.

Do mesmo modo, quando [illegible] ponto estiver na metade inferior do círculo (e, portanto, abaixo do eixo X), o cosseno será negativo.

OS GRÁFICOS DE SENO E COSSENO

Os programas a seguir mostram as mudanças do seno e do cosseno à medida que o ponto gira pelo círculo.

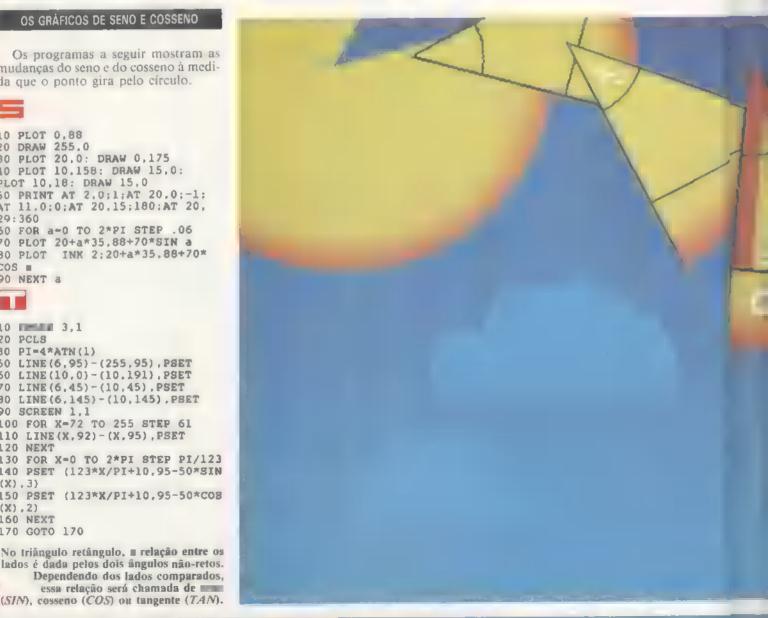
S

```
10 PLOT 0,88
20 DRAW 255,0
30 PLOT 20,0: DRAW 0,175
40 PLOT 10,158: DRAW 15,0:
PLOT 10,18: DRAW 15,0
50 PRINT AT 2,0;1;AT 20,0;-1;
AT 11,0;0;AT 20,15;180;AT 20,
29:360
60 FOR a=0 TO 2*PI STEP .06
70 PLOT 20+a*35,88+70*SIN a
80 PLOT  INK 2;20+a*35,88+70*
COS [illegible]
90 NEXT a
```

```
10 [illegible] 3,1
20 PCLS
30 PI=4*ATN(1)
50 LINE(6,95)-(255,95),PSET
60 LINE(10,0)-(10,191),PSET
70 LINE(6,45)-(10,45),PSET
80 LINE(6,145)-(10,145),PSET
90 SCREEN 1,1
100 FOR X=72 TO 255 STEP 61
110 LINE(X,92)-(X,95),PSET
120 NEXT
130 FOR X=0 TO 2*PI STEP PI/123
140 PSET (123*X/PI+10,95-50*SIN
(X),3)
150 PSET (123*X/PI+10,95-50*COS
(X),2)
160 NEXT
170 GOTO 170
```

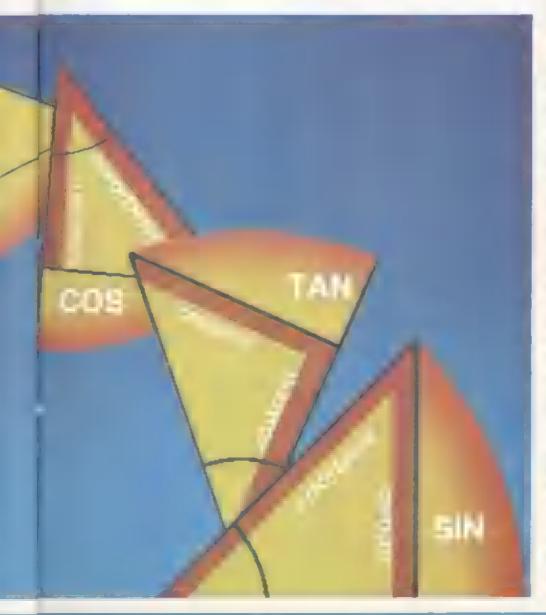
No triângulo retângulo, [illegible] relação entre os lados é dada pelos dois ângulos não-retos. Dependendo dos lados comparados, essa relação será chamada de [illegible] (*SIN*), cosseno (*COS*) ou tangente (*TAN*).

```
10 SCREEN2:COLOR1,15
20 CLS
30 PI=4*ATN(1)
40 LINE(7,96)-(256,96),1
50 LINE(11,1)-(11,192),1
60 LINE(7,46)-(11,46),1
70 LINE(7,146)-(11,146),1
[illegible] [illegible] X=72 TO 256 [illegible] 61
90 LINE(X,93)-(X,96),1
100 [illegible]
110 [illegible] X=0 TO 2*PI [illegible] PI/123
120 PSET(123*X/PI+11,96-50*SIN(
X)),6
130 PSET(123*X/PI+11,96-50*COS(
X)),10
140 [illegible]
150 IF INKEY$="" THEN 150
160 END
```

```
10  HGR2 : HCOLOR= 3
20 PI = 4 *  ATN (1)
30  HPLOT 6,95 TO 255,95
40  HPLOT 10,0 TO 10,191
[illegible]  HPLOT 6,45 TO 10,45
60  HPLOT 6,145 TO 10,145
70  [illegible] X = 72 TO 255 STEP 61
80  HPLOT X,92 TO X,95
90  NEXT X
100  FOR X = 0 TO [illegible] * PI STEP [illegible]
I / 123
110  HPLOT 123 * [illegible] / PI + 10,95
 - 50 *  SIN (X)
120  HPLOT 123 * X / PI + 10,95
 - 50 *  COS (X)
130  [illegible] [illegible]
140  GET A$: IF A$ = "" THEN 14
0
150  HOME : TEXT
```

Como ■ programa da bússola, ■ MSX, o Apple ■ o TRS-Color começam ajustando o modo gráfico adequado (linha 10). Mais uma vez, os três criam uma variável (PI) para ■ valor de PI na linha 30. Depois, todos traçam os eixos para o gráfico.

Os programas desenham pequenas linhas que representam intervalos ■ longo dos eixos. A versão para o Spectrum numera os traços. As marcas para o eixo Y (eixo vertical) são 1 e −1, ■ quais satisfazem todos ■ possíveis valores do seno e do cosseno. A numeração para o eixo X vai de 0 a 360 graus.

O computador desenha então ■ gráficos, um para ■ seno ■ outro para ■ cosseno. Ele faz isso usando um laço **FOR...NEXT**.

Inventada na China por volta de 1100, a bússola magnética só se tornou conhecida na Europa um século mais tarde. Ao lado, a bússola do Sinclair Spectrum mostra um ângulo de 70 graus.

Vimos que existem 2xPI radianos num círculo; portanto, ■ quisermos compreender todos os ângulos, o laço deverá ser **FOR T=0 TO 2*PI**. Nesse laço existem saltos (**STEP**) para que o computador não faça um desenho a cada mínima variação de ângulo, mas só ■ cada três ou quatro variações.

O **STEP** faz com que ■ programa seja mais rápido. Para ter uma idéia de sua importância, tente removê-lo e veja a diferença.

As linhas mais importantes para o cálculo dos valores do seno ■ do cosseno (140, 150 no TRS-Color; 70, 80 no TK; 120, 130 no MSX ■ 110, 120 no Apple ■ no TK-2000) parecem muito complicadas, mas na verdade são bem simples. Cada linha usa os comandos **PLOT**, **PSET** ou **DRAW** para desenhar o gráfico. As posições dos pontos são determinadas pelo seno e pelo cosseno (**SIN** e **COS**), que aparecem nessas linhas. Os outros números que os acompanham servem simplesmente para mudar os resultados de escala, a fim de ajustá-los corretamente na tela; como cada computador tem uma disposição de tela diferente, esses cálculos são diferentes para cada um.

DESENHE CÍRCULOS

A chave para posicionar caracteres ao redor de um círculo, como no programa ■ da bússola, consiste, como vimos, em conhecer o raio da circunferência ■ também os ângulos que eles formam com o eixo X.

O que precisamos é de ■ programa que forneça ■ centro e o raio do círculo ■ calcule as coordenadas X e Y de qualquer ângulo dado. Se entrarmos uma série de ângulos, ■ programa será capaz de desenhar uma série de pontos ao re-

dor do círculo escolhido. O segredo está em ensinar ■ computador a calcular as coordenadas X e Y, e é aqui que entram nossos velhos amigos **SIN** ■ **COS**.

Digite este pequeno programa:

```
20 FOR x=0 TO 2*PI STEP PI/30
30 PLOT 128+50*SIN x,88+50*
COS ■
40 NEXT ■
```

```
10 SCREEN2:COLOR1,15
20 CLS
30 PI=4*ATN(1)
40 FOR X=0 TO 2*PI STEP PI/45
50 PSET(128+80*SIN(X),96-80*COS
(X)),1
60 NEXTX
70 IF INKEY$="" THEN 70
■ END
```

```
10  HGR2 : HCOLOR= 3
20  PI = 4 ■  ATN (1)
30  FOR Z = 0 TO 80 STEP ■
40  FOR X = 0 TO ■ * PI STEP PI
 / 45
50  HPLOT 140 + Z *  SIN (X),96
 - 80 *  COS (X)
60  NEXT X
70  NEXT Z
80  GET A$: IF A$ = "" THEN 80
90  HOME : TEXT
```

A tela exibe uma série de pontos que adotam a forma de um círculo (essa forma não é definida por uma linha contínua porque tal linha levaria muito tempo para ser desenhada).

Em cada programa, exceto no do Spectrum, o computador é ajustado para o modo gráfico adequado. Em seguida, é criado um valor para PI no TRS-Color, no MSX ■ no Apple. Um laço **FOR...NEXT** diz então ao computador para percorrer uma série de ângulos, que vai de 0 a 2*PI (linha 20 no TK 90X, 30 no TRS-Color, 40 no MSX ■ no Apple) — ou seja, diz para formar um círculo completo. O tamanho do salto (**STEP**) é o responsável pela velocidade do programa e pelo visual pontilhado.

Na linha seguinte, o computador é instruído a desenhar um ponto para cada ângulo. A posição do ponto é determinada pela fórmula:

PLOT raio***SIN(X)**, raio***COS(X)**

Como você deve estar lembrado, essas funções determinam as coordenadas para qualquer ângulo dado. Os outros números somados ou subtraídos nessa linha servem para ajustar o centro do círculo em relação à tela gráfica do computador. O centro da tela de alta resolução no TRS-Color está em 127,95; no Apple, está em 140,96; no MSX, está em 128,96 e no Spectrum, está em 128,88.

Tente agora alterar a posição do centro ou o tamanho do raio: isso o ajudará a entender melhor essas funções.

Existem ainda outros números que, ao serem mudados, alteram ■ visual do círculo. Na primeira linha do laço, o **STEP** comanda a distância entre um ponto e outro. Quanto menor o **STEP**, mais contínuo será o círculo desenhado. Se quisermos maior número de pontos ■ desenho, devemos diminuir o **STEP** (ou dividir PI por um número maior).

```
10 PMODE 4,1:PCLS:SCREEN 1,1
20 PI=4*ATN(1)
30 FOR X=0 TO 2*PI STEP PI/45
40 PSET (127+80*SIN(X),95-80*CO
S(X),5)
50 NEXT ■
60 GOTO 60
```

CONSTRUA UMA ELIPSE

Uma mudança mais interessante consiste em transformar nosso círculo em uma elipse. Para tanto, basta fazermos com que o número que multiplica **SIN** seja maior ou menor que aquele que multiplica **COS**. Se o que multiplica **SIN** for maior, teremos ■ elipse baixa e larga. Se for menor, a elipse será alta e estreita.

O programa a seguir desenha uma série de elipses que, juntas, criam o efeito de um globo.

```
10 FOR z=0 TO 50 STEP 5
20 FOR x=0 TO 2*PI STEP PI/15
30 PLOT 128+z*SIN x,88+50*
COS x
40 NEXT x
50 NEXT z
```

```
10 PMODE 4,1:PCLS:SCREEN 1,1
20 PI=4*ATN(1)
30 FOR Z=0 TO 80 STEP 5
40 FOR X=0 TO 2*PI STEP PI/45
50 PSET(127+Z*SIN(X),95-80*COS(
X),5)
■ NEXT X
70 NEXT Z
80 GOTO ■
```

Qual ■ tamanho do maior círculo desenhado pelo computador?

Evidentemente, nenhum círculo desenhado pelo computador pode extravasar ■ limites de ■ tela. De fato, o raio do maior círculo deve ser, no máximo, equivalente ■ metade do menor dos lados da tela. Os maiores raios para cada computador são: ■ para o Spectrum, 95 para o TRS-Color ■ 96 para o Apple, TK-2000 e MSX.

Entretanto, para que esses números provoquem o efeito desejado, o centro do círculo deve ■ contido por eles. Se o círculo for muito grande, o Spectrum imprimirá uma mensagem de erro, enquanto ■ outros computadores desenharão o máximo que couber ■ tela.

```
10 SCREEN2:COLOR1,15
20 PI=4*ATN(1):CLS
30 FOR Z=0 TO 80 STEP 5
40 FOR X=0 TO 2*PI STEP PI/45
50 PSET(128+Z*SIN(X),96-80*COS(
X)),1
60 NEXT X
70 NEXT Z
80 IF INKEY$="" ■ 80
90 END
```

```
10  HGR2
20  HCOLOR= 3
30 PI = 4 *  ATN (1)
40  FOR X = 0 TO ■ * PI STEP PI
 / 45
50  HPLOT 140 + 80 ■  SIN (X),9
6 - ■ *  COS (X)
60  NEXT ■
70  GET A$: IF A$ = "" THEN 70
80  ■ : TEXT
```

Os programas do globo não passam de versões adaptadas do programa do círculo. Em vez de uma, apenas desenhamos agora uma série de elipses, usando um segundo laço **FOR...NEXT** para variar o número pelo qual a coordenada X é multiplicada. Isso faz com que o computador trace elipses de tamanhos diferentes.

Podemos fazer a elipse "crescer" o quanto quisermos, mudando o tamanho do **STEP** de Z (variável de controle).

MOVIMENTE FIGURAS NA TELA

- COMO CRIAR UMA FIGURA MÓVEL NA MEMÓRIA
- MOVIMENTOS MAIS RÁPIDOS
- UM SAPO QUE PULA E UM TANQUE QUE ATIRA

[illegible] é só o programador experiente que pode se divertir com linguagem de máquina. Utilize alguns programas pequenos, escritos em código, para [illegible] mais vida a [illegible] programas [illegible] BASIC.

Linguagem de máquina é coisa bem complicada. Para a maioria dos usuários de micros, ela não passa de um amontoado de números sem sentido. Porém, se você usar algumas rotinas em código dentro de programas BASIC, logo notará as vantagens da linguagem de máquina e ficará mais motivado [illegible] enfrentar as dificuldades.

Os programas que aqui apresentamos utilizam o BASIC para colocar programas em código de máquina na memória do micro. Com isto, pode-se movimentar figuras em alta resolução muito mais rapidamente que usando só BASIC.

Os programas são específicos para cada computador. O MSX e [illegible] Apple dispõem de comandos BASIC com velocidade satisfatória, e seus programas não precisam de linguagem de máquina para produzir os mesmos resultados dos outros micros. Já publicamos um programa desse tipo para [illegible] ZX-81.

Para montar os gráficos das figuras 1 e 2, precisamos fazer três coisas. Primeiro, criar na memória do Spectrum [illegible] "grade" ou quadriculado que conterá o desenho em alta resolução. Esse quadriculado será composto por *caracteres definidos pelo usuário*. Segundo, elaborar um programa que movimente o desenho na tela. Terceiro, trocar os caracteres gráficos [illegible] quadriculado pela figura que queremos mover — no nosso caso, o tanque e o sapo.

CARACTERES DEFINIDOS PELO USUÁRIO

O Spectrum permite ao usuário mudar o formato de determinados caracteres (caracteres definidos pelo usuário, ou UDGs). Cada caractere é formado por 64 pontos — alguns com a cor do fundo (INK) e alguns com a cor da frente (PAPER) —, num arranjo de oito por oito. Desde que não ultrapassemos os limites deste quadrado, podemos escrever um programa que dê ao caractere o formato que desejarmos.

Existem 21 caracteres desse tipo: A até U, inclusive. Como mostra [illegible] figura 1, é possível juntar os caracteres — conjuntos de oito por oito pontos — para formar figuras maiores. Pode-se ter, por exemplo, duas figuras com três por três caracteres [illegible] mesmo tempo (ainda sobrarão 2 UDGs), [illegible] cinco figuras com dois por dois caracteres.

Tanto o sapo quanto o tanque apresentados neste artigo usam uma figura de três por três caracteres, mais ou menos assim:

```
A  B  C

D  E  F

G  H  I
```

Pode-se colocar isto na tela empregando **PRINT AT**, em BASIC; porém, a melhor alternativa é usar a rotina em linguagem de máquina criada pelo programa a seguir.

```
 10 IF [illegible] 23733=127 THEN
CLEAR 32399: LET B=32400:  LET
Z=0
 20 IF [illegible] 23733=255 [illegible]
CLEAR 65199: LET B=65200: LET
Z=1
 30 [illegible] [illegible] TO B+129: [illegible] A:
POKE N,A: NEXT N
 40 IF Z=1 THEN  POKE 65258,
178: POKE 65259,254: [illegible]
65277,179: POKE 65278,254
 50 SAVE "Padrao"CODE B,130
 60 STOP
100 DATA 24,55,1,22,0,0,32,32,
32,22,0,0,32,32,32,22,0,0,32,
32
110 DATA 32,22,0,0,144,145,146
,22,0,0,147,148,149,22,0,0,150
,151,152,22
120 DATA 0,0,153,154,155,22,0,
0,156,157,158,22,0,0,159,160,
161,58,146,126
130 DATA 254,1,1,18,0,40,8,56,
4,203,33,24,2,14,0,221,33,147,
126,221
140 DATA 9,58,137,92,71,62,24,
144,221,119,1,60,221,119,7,60,
221,119,13,58
150 DATA 136,92,71,62,33,144,
221,119,2,221,119,8,221,119,14
,221,229,62,2,205
160 DATA 1,22,209,1,18,0,205,
60,32,201
```

Talvez você ache que esta é uma maneira complicada de fazer o que alguns poucos **PRINT AT** fariam. Mas existem algumas razões para a substituição:
1. Uma vez digitado [illegible] gravado, este pro-

1. Desenho do tanque. Cada quadrado grande tem oito por oito pontos.

grama pode [illegible] usado quantas vezes você quiser.

2. Quando chamada de dentro de um programa BASIC, a rotina criada imprimirá os caracteres [illegible] velocidade muito maior que os **PRINT AT**.

Sem saber código de máquina é impossível compreender o que os números de cada linha **DATA** significam, [illegible] tentaremos dar uma idéia de como funciona o programa em BASIC.

As linhas 10 e 20 determinam onde a rotina em código será colocada, dependendo do tamanho da memória de [illegible] Spectrum — 16K ou 48K. Note que os comandos podem não funcionar em micros de 16K com expansão. Se este for o caso, mude a linha 20 para:

```
20 CLEAR 65199: LET B=65200:
LET Z=1
```

Os números incluídos nas linhas **DATA** são transferidos para [illegible] memória do computador pela linha 30.

A linha 50 grava o programa. Você pode mudar o nome dele, se quiser.

As linhas 100 a 160 contêm os números que, quando colocados na memória, formam [illegible] rotina em código.

Esta rotina é gravada de uma maneira diferente. Após digitar e rodar [illegible] programa, o computador pedirá [illegible] usuário que ligue o gravador [illegible] aperte qualquer tecla, a fim de gravar a rotina em linguagem de máquina.

Já que a rotina em código está dentro da memória — colocada pelo programa BASIC ou lida por meio do cassete —, vamos usá-la. Pressione **NEW** [illegible] **ENTER**, para que o programa antigo não atrapalhe o novo.

```
 20 LET print=32400: LET B=
32402: IF PEEK 23733=255 [illegible]
LET print=65200: LET B=65202
 90 BORDER 7: [illegible] 7: INK 4:
CLS
100 LET Y=8: LET X=15: LET Y1=
8: LET X1=15: LET Z=1
110 LET A$=INKEY$
120 IF A$="z" [illegible] X>0 THEN
LET X1=X-1: LET Z=1
130 IF A$="x" AND X<29 [illegible]
LET X1=X+1: LET Z=2
140 IF A$="p" [illegible] Y>0 THEN
LET Y1=Y-1
150 IF A$="l" AND Y<18 [illegible]
LET Y1=Y+1
170 LET X=X1: LET Y=Y1
180 PRINT AT Y,X;: [illegible] B,Z:
RAND USR print
190 GOTO 110
```

Note que [illegible] palavra "print" em minúsculas deve ser digitada letra por letra, [illegible] contrário de **PRINT**, que fica na tecla **P**.

Este programa permite [illegible] movimentação de figuras [illegible] tela usando as teclas Z, X, P e L. Quando ele for rodado, veremos que foram criadas duas figuras com três por três caracteres; a segunda é formada por JKLMNOPQR. Pode-se, assim, obter duas versões do mesmo gráfico. No caso do tanque, um aponta pa[illegible] a direita, quando vai nesta direção, e outro aponta para a esquerda, quando se movimenta para a esquerda.

Veremos também que, conforme a figura se move, deixa para trás um rastro que não foi planejado. Para corrigir isto, acrescente a linha:

```
[illegible] PRINT AT Y,X;: POKE B,0:
[illegible] [illegible] print
```

Ela produz [illegible] figura de três por três caracteres [illegible] branco, que vai apagando [illegible] posições já ocupadas.

UM TANQUE DE GUERRA

Para transformar os caracteres na figura de um tanque, acrescente as próximas linhas. Elas montam o tanque apontando para [illegible] direita no quadriculado 1 [illegible] para a esquerda no quadriculado 2.

```
  10 [illegible] [illegible] "a" TO USR "r"+7
: [illegible] A: POKE N,A: NEXT [illegible]
1000 [illegible] 0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0
1010 DATA 0,0,0,0,0,0,0,0,0,0,0
,0,255,0,1,0
1020 DATA 0,0,1,63,255,255,255,
0,0,0,192,224,254,254,224,0
1030 [illegible] 63,127,255,122,48,6,0
,0,255,255,255,235,65,102,0,0
1040 DATA 255,255,255,174,6,100
,0,0
1050 DATA 0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0
1060 DATA 0,0,0,0,0,0,0,0,0,0,3
,7,127,127,7,0
1070 DATA 0,0,128,252,255,255,2
55,0,0,0,0,0,255,0,128,0
1080 [illegible] 255,255,255,117,96,38
,0,0,255,255,255,215,130,102,0,
0
1090 DATA 252,254,255,94,12,96,
0,0
```

Resta criar [illegible] dois caracteres que irão "armar" [illegible] tanque. Aperte **BREAK** e digite:

```
 10 FOR [illegible] "a" TO USR "t"+7
: [illegible] A: POKE N,A: [illegible] N
115 IF INKEY$=" " THEN  GOTO
200
200 IF Z=2 THEN  GOTO 300
```

```
210 FOR N=X-1 TO 0 STEP -1
220 [illegible] [illegible] 5;AT Y+1,N;CHR$
162
230 PAUSE 1
240 PRINT [illegible] Y+1,N;" "
250 NEXT N
260 GOTO 110
300 FOR N=X+3 TO 31
310 PRINT [illegible] 5;AT Y+1,N;CHR$
163
320 PAUSE 1
330 PRINT AT Y+1,N;" "
340 [illegible] [illegible]
350 GOTO 110
1100 DATA 0,4,9,2,176,2,9,4,0,3
2,144,64,13,64,144,32
```

Agora a tecla **SPACE** pode ser usada para dar tiros.

UM SAPO

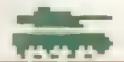
Para transformar as figuras em um sapo, você precisará livrar-se do tanque. Para isso, grave [illegible] programa e depois pressione **<NEW>** e **<ENTER>**.

O tanque [illegible] o programa que o movimentava serão anulados, mas [illegible] rotina em código que cuida do quadriculado permanecerá na memória até que [illegible] micro seja desligado.

Digite **CLEAR 32399** no Spectrum de 16K, ou **CLEAR 65199**, no de 48K, reservando o topo da memória para a rotina em código. Digite, depois, **LOAD "" CODE** e ligue o gravador para carregar a rotina em código do cassete.

Com a rotina na memória do micro, digite e rode o programa que cria o sapo.

```
30  RESTORE 5000: FOR N=USR "
a" TO USR "r"+7: [illegible] A: POKE N
,A: NEXT [illegible]
5000 DATA 0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0
5010 DATA 0,0,0,0,0,0,0,0,0,0,0
,0,0,0,1,1
5020 DATA 0,0,0,0,0,128,192,176
,0,0,0,0,0,0,0,0
5030 DATA 4,15,31,63,127,254,24
8,127,96,240,224,192,64,32,156,
192
5040 DATA 0,0,0,0,0,0,0,0
5050 DATA 0,0,0,0,0,0,0,0,0,0,0
,0,0,1,3,7
5060 DATA 8,28,27,70,255,254,25
2,249,0,0,0,0,0,0,0,0
5070 DATA 7,7,15,30,54,38,70,70
,250,196,0,0,0,0,0,0
5080 [illegible] 0,0,1,1,3,6,2,0,140,1
44,16,32,32,48,32,0
5090 DATA 0,0,0,0,0,0,0,0
```

Para controlar o movimento do sapo, digite:

```
10 [illegible] 0: [illegible] 0: INK 4:
BRIGHT 1: CLS
20 LET P=32400: IF [illegible] 23733
=255 [illegible] LET P=65200
100 PRINT AT 10,0;: RAND USR P
: IF INKEY$="" THEN  GOTO 100
110 RESTORE 1000: FOR F=1 TO 5
120 READ A,B,C: POKE P+2,A:
PRINT AT B,C;: RAND USR P
130 PAUSE 2: CLS : NEXT [illegible]
150 PRINT AT 10,12;: RAND USR
P: IF INKEY$="" THEN  GOTO 150
200 FOR F=1 TO 5
210 READ A,B,C: POKE P+2,A:
PRINT AT B,C;: RAND [illegible] P
220 PAUSE 2: CLS : NEXT F
230 GOTO 100
1000 DATA 1,10,0,2,7,3,2,5,6,2,
7,9,1,10,12,1,10,12,2,7,15,2,5,
18,2,7,21,1,10,24
```

O Apple não precisa de linguagem de máquina para movimentar uma figura em alta resolução: basta criar [illegible] tabela de figuras na memória do micro e usar o comando **DRAW**.

Se não dispusermos de um editor de figuras, o trabalho poderá ser bem complicado. Mas você não será obrigado a usá-lo, pois o programa a seguir cria o tanque e [illegible] movimenta.

```
10   HOME :E = 35000: HIMEM: E
20 F =  INT (E / 256): POKE 232
,E - F * 256: POKE 233,F
30   FOR I = E TO [illegible] + 41 + 14 *
32
40   READ A: POKE I,A
50   NEXT
60   HGR : SCALE= 1: ROT= 0:X =
130:Y = 90:F = 0: GOTO 410
70 LX = X:LY = Y:LF = F
80   GET K$: IF K$ = "" THEN 80
90   IF K$ = "P" AND Y > 10 THEN
Y = Y - 2: GOTO 140
100  IF K$ = "L" [illegible] Y < 140 TH
EN Y = Y + 2: GOTO 140
110  IF K$ = "Z" AND [illegible] > 12 THE
N [illegible] = X - 3:F = 0: GOTO 140
120  IF K$ = "X" AND X < 220 [illegible]
[illegible] X = X + 3:F = 1: GOTO 140
130  IF K$ = " " THEN 490
140  IF LX = X AND LY = Y THEN
80
150  IF LF = 0 THEN 250
160  HCOLOR= 0
170  DRAW 1 AT LX,LY
180  DRAW 2 AT LX,LY + 8
190  DRAW 3 AT LX + 8,LY
200  DRAW 4 AT LX + 8,LY + 8
210  DRAW 5 AT LX + 16,LY
220  [illegible] 6 AT LX + 16,LY + 8
230  IF F = 0 THEN 410
240  GOTO 330
250  HCOLOR= 0
260  DRAW 9 AT LX,LY
270  DRAW 10 AT LX,LY + 8
280  DRAW 11 AT LX + 8,LY
290  DRAW 12 AT LX + 8,LY + 8
300  DRAW 13 AT LX + 16,LY
310  DRAW 14 AT LX + 16,LY + 8
320  IF F = 0 THEN 410
330  HCOLOR= 3
340  [illegible] 1 AT X,Y
350  DRAW [illegible] AT X,Y + 8
360  DRAW 3 AT X + 8,Y
370  DRAW 4 AT X + 8,Y + [illegible]
380  DRAW 5 AT X + 16,Y
390  DRAW 6 AT X + 16,Y + 8
400  GOTO 70
410  HCOLOR= 3
420  DRAW 9 AT X,Y
430  DRAW 10 AT X,Y + 8
440  DRAW 11 AT X + 8,Y
450  DRAW 12 AT [illegible] + 8,Y + 8
460  DRAW 13 AT X + 16,Y
470  DRAW 14 AT X + 16,Y + 8
480  GOTO 70
490  IF F = 0 THEN 560
500  HCOLOR= 3
510  DRAW 7 AT X + 24,Y
520  FOR I = 1 TO 100: NEXT
530  HCOLOR= 0
540  DRAW 7 AT X + 24,Y
550  GOTO 80
560  HCOLOR= 3
570  DRAW [illegible] AT X - 8,Y
580  FOR I = 1 TO 100: NEXT
590  HCOLOR= 0
600  DRAW 8 AT X - 8,Y
610  GOTO 80
2000  DATA  20 ,0 ,42 ,0 ,74 ,0
,106 ,0 ,138 ,0 ,170 ,0 ,202 ,
0 ,234 ,0 ,10 ,1 ,42 ,1 ,74 ,1
,106 ,1 ,138 ,1 ,170 ,1 ,202 ,1
,234 ,1 ,10 ,2 ,42 ,2 ,74 ,2 ,
106 ,2 ,138 ,2
2010  DATA   72 ,73 ,73 ,218 ,
219 ,219 ,74 ,73 ,41 ,213 ,63 ,
223 ,155 ,41 ,45 ,45 ,173 ,59 ,
63 ,63 ,191 ,73 ,9 ,45 ,213 ,21
[illegible] ,219 ,19 ,0 ,0 ,0,0
2020  DATA   40 ,45 ,45 ,45 ,
213 ,63 ,63 ,63 ,55 ,45 ,45 ,45
,173 ,251 ,31 ,63 ,87 ,109 ,73
,209 ,59 ,223 ,159 ,73 ,73 ,13
7 ,219 ,219 ,155 ,0 ,0 ,0
2030  DATA   72 ,73 ,73 ,218 ,
219 ,219 ,106 ,73 ,73 ,218 ,59
,63 ,63 ,46 ,45 ,45 ,45 ,213 ,6
[illegible] ,63 ,63 ,55 ,45 ,45 ,45 ,173
,219,219 ,155 ,0 ,0, 0
2040  DATA   40 ,45 ,45 ,45 ,
213 ,63 ,63 ,63 ,55 ,45 ,45 ,45
,173 ,59 ,255 ,31 ,55 ,77 ,73
,141 ,27 ,255 ,59 ,87 ,73 ,73 ,
209 ,219 ,219 ,19 ,0 ,0
2050  DATA   72 ,73 ,73 ,218
,219 ,219 ,74 ,73 ,73 ,218 ,219
,219 ,42 ,45 ,45 ,45 ,213 ,219
,219 ,19 ,77 ,73 ,137 ,219 ,21
9 ,155 ,0 ,0 ,0 ,0 ,0 ,0
2060  DATA   40 ,45 ,45 ,77 ,
218 ,63 ,63 ,63 ,46 ,45 ,45 ,45
,213 ,59 ,63 ,31 ,87 ,73 ,109
,209 ,219 ,27 ,191 ,73 ,73 ,137
```

```
.219 .219 .155 .0 .0 .0
2070  DATA    72 .73 .73 .218
.219 .251 .106 .105 .73 .218 .2
19 .27 .87 .73 .109 .213 .219 .
219 .23 .77 .77 .137 .219 .219
.159 .0 .0 .0 .0 .0 .0 .0
2080  DATA    72 .73 .73 .218
.251 .219 .74 .9 .77 .213 .251
.219 .19 .13 .109 .73 .218 .223
.219 .74 .9 .77 .213 .27 .223
.155 .0 .0 .0 .0 .0 .0
2090  DATA    72 .73 .73 .218
.219 .219 .74 .73 .73 .218 .219
.219 .42 .45 .45 .45 .213 .219
.219 .83 .73 .73 .213 .219 .21
9 .19 .0 .0 .0 .0 .0 .0
2100  DATA    72 .45 .45 .173
.59 .63 .63 .191 .45 .45 .45 .1
73 .27 .31 .63 .191 .9 .109 .73
.218 .255 .219 .74 .73 .73 .21
8 .219 .219 .2 .0 .0. ■
2110  DATA    72 .73 .73 .218
.219 .219 .74 .73 .9 .213 .63 .
63 .255 .42 .45 .45 .45 .213 .6
3 .63 .63 .55 .45 .45 .45 .173
.219 .219 .155 .0 .0 .0
2120  DATA    40 .45 .45 .45 .
213 .63 .63 .63 .55 .45 .45 .45
.173 .59 .31 .31 .63 .14 .77 .
73 .213 .59 .223 .191 .73 .73 .
137 .219 .219 .155 .0. 0
2130  DATA    72 .73 .73 .218
.219 .219 .42 .77 .73 .209 .219
.27 .63 .46 .45 .45 .109 .218
.63 .63 .63 .46 .109 .73 .209 .
219 .219 .19 .0 .0 .0 .0
2140  DATA    40 .45 .45 .45 .
213 .63 .63 .63 .55 .45 .45 .45
.173 .27 .63 .31 .31 .78 .73 .
109 .218 .251 .59 .87 .73 .73 .
209 .219 .219 .19 .0 .0
```

A linha 10 reserva o topo da memória para a tabela de figuras. O Apple é então informado da localização da mesma pela linha 20. O **FOR...NEXT** das linhas 30 a 50 monta a tabela na memória.

A linha 60 liga a tela de alta resolução, estabelece a escala e a orientação do desenho. Além disso, coloca as coordenadas do centro da tela em **X** ■ **Y**. Em seguida, o programa é desviado para a linha 140, para que uma imagem inicial do tanque seja feita.

As linhas 70 a 140 movimentam o tanque através das teclas Z, X, P e L. Tratam também de não deixá-lo ultrapassar os limites da tela. As variáveis **F** ■ **LF** são sinalizadores que indicam a direção atual e antiga do tanque; desenham e apagam a figura ■ orientação correta. A linha 140 evita que o tanque seja apagado e desenhado, sem sair do lugar. A linha 130 verifica se a tecla de espaço foi pressionada, saltando para a linha 420, onde começa ■ porção do programa que dispara um tiro — na direção correta!

As linhas 150 a 320 apagam o tanque de sua última posição, conforme o valor de **LF**. As linhas 330 a 480 desenham-no na nova posição, conforme o valor de F.

Note que as linhas **DATA** 2000 a 2140 foram geradas com o programa editor apresentado no artigo da página 316. Os blocos do tanque são numerados de cima para baixo e, depois, da esquerda para a direita.

HCOLOR = 3 desenha o tanque em branco. Se utilizarmos **HCOLOR = 1**, teremos a cor verde, mas o desenho será ligeiramente deformado. Isto ocorre devido ■ maneira como o Apple usa as cores em alta resolução.

UM SAPO

O programa a seguir monta uma tabela correspondente ao sapo da figura 2.

```
10   HOME :E = 35000: HIMEM: E
20   POKE 233, INT (E / 256): PO
■■ 232,E - 256 *  PEEK (233)
30   FOR I = E TO E + 41 + 10 *
32: READ A: POKE I,A: NEXT I
40   ■■■ : SCALE= 1: ROT= 0
100   GOSUB 1800: GOSUB 1300
110   GOSUB 1000
120   GET K$: IF K$ <  > " " THE
N 120
130   GOSUB 1200: GOSUB 1000
140   GOSUB 1500: GOSUB 1300
150   GOSUB 1040: GOSUB 1400
160   GOSUB 1200: GOSUB 1500
170   GOSUB 1040: GOSUB 1300
180   GOSUB 1600: GOSUB 1040
190   GOSUB 1400: GOSUB 1200
200   GOSUB 1600: GOSUB 1040
210   GOSUB 1300: GOSUB 1700
220   GOSUB 1000: GOSUB 1400
230   GOSUB 1200: GOSUB 1700
240   GOSUB 1000: GOTO 100
1000   DRAW 1 AT X,Y
1010   DRAW 2 AT X,Y + ■
1020   ■■■■ 3 AT X + 8,Y
1030   DRAW 4 AT X + 8,Y + 8
1035   RETURN
1040   DRAW 5 AT X,Y
1050   DRAW 6 AT ■ + 8,Y - 16
1060   DRAW 7 AT X + 8,Y - 8
1070   DRAW 8 AT X + 8,Y
1080   ■■■■ ■ AT X + 16,
Y - 16
1090   DRAW 10 AT X + 16,
Y - 8
1100   RETURN
1200   HCOLOR= 0:
RETURN
1300   HCOLOR= 3:
RETURN
1400   FOR I = 1
TO 200: NEXT I
1410   RETURN
1500 X = 90:Y =
50: RETURN
1600 X = 160:Y =
50: RETURN
1700 X = 230:Y =
90: RETURN
1800 X = 30:Y = 90: RETURN
2000  DATA   20 .0 .42 .0 .74 .
0 .106 .0 .138 .0 .170 .0 .202
.0 .234 .0 .10 .1 .42 .1 .74 .1
.106 .1 .138 .1 .170 .1 .202 .
1 .234 .1 .10 .2 .42 .2 .74 .2
.106 .2 .138 .2
2010  DATA    0 .72 .73 .73 .21
8 .219 .219 .74 .73 .73 .218 .2
19 .219 .74 .73 .73 .218 .219 .
219 .74 .73 .9 .213 .223 .219 .
19 .0 .0 .0 .0 .0 .0
2020  DATA    0 .72 .73 .77 .26
.63 .255 .155 .73 .45 .45 .213
.63 .63 .255 .10 .45 .45 .45 .
213 .59 .63 .63 .55 .45 .45 .77
.209 .63 .63 .63 .23
2030  DATA    0 .72 .73 .73 .21
8 .219 .219 .74 .73 .73 .218 .2
19 .219 .74 .73 .73 .218 .219 .
219 .46 .77 .73 .209 .219 .59 .
31 .6 .0 .0 .0 .0 .0
2040  DATA    0 .8 .109 .73 .20
9 .219 .59 .63 .46 .109 .73 .20
9 .219 .219 .119 .77 .73 .209 .
219 .27 .159 .73 .45 .77 .218 .
219 .219 .2 .0 .0 .0 .0
2050  DATA    0 .72 .73 .73 .21
8 .219 .219 .74 .73 .73 .26 .22
3 .219 .83 .73 .9 .173 .27 .255
.219 .74 .73 .105 .218 .219 .2
19 .2 .0 .0 .0 .0 .0
2060  DATA    0 .72 .73 .73 .21
■ .219 .219 .74 .73 .73 .218 .2
19 .219 .74 .73 .73 .26 .223 .2
19 .83 .73 .9 .173 .59 .255 .21
■ .2 .0 .0 .0 .0 .0
2070  DATA    0 .72 .73 .45 .21
3 .63 .223 .155 .73 .41 .45 .21
3 .59 .63 .223 .74 .109 .109 .2
18 .255 .251 .10 .77 .41 .141 .
27 .255 .27 .23 .0 .0 .0
2080  DATA    0 .104 .9 .109 .2
09 .219 .251 .51 .77 .77 .137 .
219 .219 .159 .9 .77 .73 .218 .
219 .255 .74 .77 .73 .218 .219
```

```
,219 ,2 ,0 ,0 ,0 ,0 ,0
2090  DATA    0 ,72 ,9 ,77 ,209
 ,27 ,63 ,223 ,74 ,41 ,13 ,173
,27 ,255 ,27 ,23 ,45 ,45 ,45 ,1
73 ,27 ,63 ,63 ,63 ,46 ,45 ,45
,77 ,26 ,223 ,63 ,63
2100  DATA    0 ,40 ,45 ,109 ,1
41 ,219 ,223 ,63 ,78 ,73 ,73 ,2
18 ,219 ,219 ,74 ,73 ,73 ,218 ,
219 ,219 ,74 ,73 ,73 ,218 ,219
,219 ,2 ,0 ,0 ,0 ,0 ,0
```

Use a barra de espaço para fazer o sapo pular.

T

Para definir ■ mover os desenhos das figuras 1 ■ 2, precisamos fazer três coisas. Primeiro, criar ■ memória do computador uma "grade" ■ quadriculado onde possamos desenhar ■ que quisermos. Este quadriculado será composto por *caracteres definidos pelo usuário*. Segundo, transferir o padrão do desenho desejado — representado por números em linha **DATA** — para o quadriculado. Terceiro, elaborar um programa ■ BASIC que desenhe e movimente o tanque ■ o sapo.

Um caractere definido pelo usuário (UDG) é um "quadro", dentro do qual se coloca parte de um desenho em alta resolução. O quadro é composto por 64 pontos dispostos num arranjo de oito por oito pontos. Cada ponto pode ■ preto ou branco em **PMODE 4,1** (este é o **PMODE** mais flexível; em outros é preciso definir dois ou quatro pontos de cada vez).

Se colocarmos vários desses caracteres lado a lado, poderemos construir desenhos maiores e mais detalhados. À diferença de outros computadores, ■ TRS-Color não possui UDGs embutidos. Mas porções de ■ memória se comportarão como tal, ■ fizermos um programa para isto.

Teoricamente, dispomos de espaço para criar muitos caracteres, ■ seria trabalhoso usar vários deles. Na prática, quatro ■ cinco bastam.

COMO CRIAR O QUADRICULADO

Para produzir o tanque da figura 1, ou ■ sapo da figura 2, precisamos primeiro fazer com que uma porção da memória se comporte como UDGs. Os dois desenhos requerem um quadriculado de três por três caracteres — 24 por 24 pontos.

Se usássemos o BASIC para desenhar uma figura destas, precisaríamos de um **PSET** para desenhar cada ponto, ou seja, 576 **PSET** ao todo. Assim, um programa em linguagem de máquina será mais rápido.

Digite:

```
10 CLEAR 200,32000
20 ■ I=32000 TO 32110
30 READ N
40 ■ I,N
50 NEXT
60 CLS
90 PRINT "APERTE  QUAL
QUER   TECLA  PARA
■ ■ PROGRAMA EM
LINGUAGEM  DE MAQUI
NA"
100 B$=INKEY$
11' IF B$="" THE
N 100
120 ■ "P
RPADRAO",
32000,321
10,32000
130 DATA 190,127,188,134,3,183,
125,111,183,125,112,134,8,183,1
25,113
140 DATA 182,125,250,39,50,206,
126,44,74,198,72,61,51,203,166,
192
150 ■ 167,132,48,136,32,122,
125,113,38,244,134,8,183,125,11
3,48
160 DATA 137,255,1,122,125,111,
38,230,134,3,183,125,111,48,137
,0
170 DATA 253,122,125,112,38,216
,57,95,231,132,48,136,32,122,12
5,113
180 DATA 38,246,134,8,183,125,1
13,48,137,255,1,122,125,111,38,
232
190 DATA 134,3,183,125,111,48,1
37,0,253,122,125,112,38,218,57
```

Este programa cria um outro, em linguagem de máquina. Sua gravação deve ■ feita pelo próprio programa; portanto, tenha o gravador por perto — os programas aqui apresentados não funcionam com o drive de disquetes conectado.

Além de produzir desenhos em alta resolução numa velocidade muito maior, o programa em código ocupa bem menos memória que ■ rotina BASIC.

Se você conhece a linguagem de máquina do 6809, não entenderá o que os números nas linhas **DATA** significam, mas pode ter uma idéia de como funciona o programa em BASIC.

A linha 10 reserva a memória necessária para acomodar o programa ■ código. As linhas 90 a 120 transferem ■ programa da memória para a fita.

UM TANQUE DE GUERRA

Para desenhar ■ tanque são necessárias duas figuras de três por três caracteres — uma para o tanque apontando para ■ direita ■ outra para o tanque que aponta para a esquerda.

Digite **NEW** para anular o programa antigo — ■ programa em código permanecerá na memória. Este novo programa cria ■ tanque mas, por enquanto, não ■ desenha.

```
10 CLEAR 200,32000
20 ■ I=32300 TO 32443
30 ■ N
40 POKE I,N
50 NEXT
60 DATA 0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0
70 ■ 0,0,3,7,127,127,7,0,0,0
,128,252,255,255,255,0
■ DATA 0,0,0,0,255,0,128,0,255
,255,255,117,96,38,0,0
■ DATA 255,255,255,215,130,102
,0,0,252,254,255,94,12,96,0,0
100 ■ 0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0
110 ■ 0,0,0,0,255,0,1,0,0,0,
```

```
1,63,255,255,255,0
120 DATA 0,0,192,224,254,254,22
4,0,63,127,255,122,48,6,0,0
130 DATA 255,255,255,235,65,102
,0,0,255,255,255,174,6,100,0,0
```

Para desenhar ■ mover o tanque, acrescente as próximas linhas ao programa anterior:

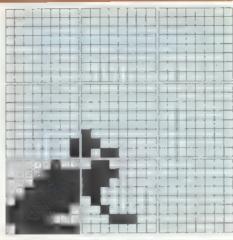
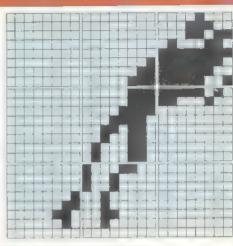
```
5 PCLEAR 5
170 PMODE 4,1
180 PCLS
290 PCLS
300 SCREEN 1,1
310 T=1
320 TP=3500
330 POKE 32700,INT(TP/256)
340 POKE 32701,TP-256*INT(TP/25
6)
350 POKE 32250,T
360 EXEC 32000
370 LP=TP
380 IF PEEK(338)=251 THEN TP=TP
-32:GOTO 440
390 IF PEEK(342)=253 THEN TP=TP
+32:GOTO 440
400 IF PEEK(340)=247 THEN TP=TP
-1:T=2:GOTO 440
410 IF PEEK(338)=247 THEN TP=TP
+1:T=1:GOTO 440
430 GOTO 380
440 IF TP<1536 OR TP>6941 THEN
TP=LP
470 GOTO 330
```

Agora digite **RUN**; usando as teclas P, L, Z e X, você poderá mover ■ tanque. Este deixará um rastro atrás de si. Para evitar o problema, digite:

```
450 POKE 32250,0
460 EXEC 32000
```

Estas linhas usam o programa em código para imprimir uma figura composta por caracteres em branco.

Para que o canhão atire, acrescente mais algumas linhas ■ programa. Você poderá fazê-lo por meio da barra de espaço.

```
160 DIM A(2),B(2),C(2)
190 FOR I=1536 TO 1760 STEP 32
190 FOR I=1536 TO 1760 STEP 32
200 READ N
210 POKE I,N
220 NEXT
230 GET (0,0)-(7,7),A
240 ■ I=1536 TO 1760 STEP 32
250 READ ■
260 POKE I,N:NEXT
280 GET (0,0)-(7,7),B
420 IF PEEK(345)=247 GOSUB 500
480 DATA 0,32,144,64,13,64,144,
32
490 DATA 0,4,9,2,176,2,9,4
500 IF T=2 THEN 560
510 YP=INT((TP-1536)/32)+8
520 XP=8*(TP-1533-(YP-8)*32)
530 IF XP>255 THEN 620
540 PUT (XP,YP)-(XP+7,YP+7),A
550 GOTO 600
```

2. Desenho do sapo. A figura pula quando se pressiona a tecla de espaço.

```
560 YP=INT((TP-1536)/32)+8
570 XP=8*(TP-1537-(YP-8)*32)
580 IF XP<0 THEN 620
590 PUT (XP,YP)-(XP+7,YP+7),B
600 FOR I=1 TO 100:NEXT
610 PUT (XP,YP)-(XP+7,YP+7),C
620 RETURN
```

UM SAPO

A figura 2 mostra o desenho de um sapo. Depois que ele for transferido para o quadriculado da memória e puder ser controlado pelo programa em BASIC, sairá pulando tela afora.

Para transferir o desenho do sapo para ■ quadriculado, utilize o programa ■ seguir. Antes, porém, use **NEW** para limpar a memória ocupada pelo BASIC.

```
10 CLEAR 200,32000
20 FOR I=32300 TO 32443
30 READ N
40 ■ I,N:NEXT
60 DATA 0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,■
70 DATA 0,0,0,0,0,0,1,1,0,0,0,0
,0,128,192,176
80 DATA 0,0,0,0,0,0,0,0,4,15,31
,63,127,254,248,127
90 DATA 96,240,224,192,64,32,15
6,192,0,0,0,0,0,0,0,0
100 DATA 0,0,0,0,0,0,0,0,0,0,0,
0,0,1,3,7,8,28,27,70,255,254,25
2,249
110 DATA 0,0,0,0,0,0,0,0,7,7,15
,30,62,54,70,70
120 DATA 250,196,0,0,0,0,0,0,0,
0,1,1,3,6,2,0
130 DATA 140,144,16,32,32,48,32
,0,0,0,0,0,0,0,0,0
```

Para fazer o sapo pular, digite **NEW** e, em seguida, este programa:

```
10 PCLEAR 5
20 PMODE 4,1
30 PCLS
40 SCREEN 1,1
50 FP=4487
60 ■ 32700,17
70 ■ 32701,135
80 POKE 32250,1
90 EXEC 32000
100 IF PEEK(345)<>247 THEN 100
110 RESTORE
120 ■ I=1 TO 5
130 READ F,FD
140 ■ 32700,INT(FP/256)
150 POKE 32701,FP-256*INT(FP/25
6)
160 ■ 32250,F
170 EXEC 32000
180 FOR J=1 TO 70:NEXT
190 POKE 32250,0
200 ■ 32000
210 FP=FP+FD
220 ■
230 GOTO 80
240 DATA 1,-287,2,-253,2,258,2,
291,1,0
```

Os números da linha **DATA** no final do programa fazem o sapo descrever um arço quando pressionamos a barra de espaço.

MSX

O BASIC do MSX dispensa o uso de linguagem de máquina, pois dispõe de comandos gráficos poderosos e é suficientemente rápido para criar e movimentar figuras em alta resolução.

O programa a seguir cria e movimen■ o tanque da figura 1.

```
■ CLEAR 300:COLOR 12,15,12
10 SCREEN 1,3:KEY OFF
20 FOR I=1 TO 16*8
30 READ A:A$=A$+CHR$(A)
40 NEXT I
50 SPRITE$(0)=LEFT$(A$,32)
60 SPRITE$(1)=MID$(A$,33,32)
70 SPRITE$(2)=MID$(A$,65,32)
80 SPRITE$(3)=MID$(A$,97,32)
90 SPRITE$(4)=MID$(A$,33,16)
100 SPRITE$(5)=STRING$(16,0)+MI
D$(A$,81,16)
110 X=100:Y=90:F=2
111 PUT SPRITE 0,(X-16,Y),12,5
112 PUT SPRITE 1,(X+16,Y),12,3
```

```
120 K$=INKEY$:IF K$="" THEN 120
130 IF ASC(K$)=29 AND X>0 THEN X=X-1:F=2:GOTO 200
140 IF ASC(K$)=28 AND X<210 THEN X=X+1:F=1:GOTO 200
150 IF ASC(K$)=30 AND Y>0 THEN Y=Y-1:GOTO 200
160 IF ASC(K$)=31 AND Y<160 THEN Y=Y+1:GOTO 200
200 ON F GOSUB 500,600
210 GOTO 120
500 PUT SPRITE 0,(X,Y),12,0
510 PUT SPRITE 1,(X+32,Y),12,4
520 RETURN
600 PUT SPRITE 0,(X-16,Y),12,5
610 PUT SPRITE 1,(X+16,Y),12,3
620 RETURN
5000 DATA  0 , 0 , 3 , 7 , 127 , 127 , 7 , 0 , 255 , 255 , 255 , 117 , 96 , 38 , 0 , 0 , 0 , 0 , 128 , 252 , 255 , 255 , 255 , 0 , 255 , 255 , 255 , 215 , 130 , 102 , 0 , 0
5010 DATA  0 , 0 , 0 , 0 , 255 , 0 , 128 , 0 , 252 , 254 , 255 , 94 , 12 , 96 , 0 , 0 , 0 , 32 , 144 , 64 , 13 , 64 , 144 , 32 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
5020 DATA  0 , 4 , 9 , 2 , 176 , 2 , 9 , 4 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 255 , 0 , 1 , 0 , 63 , 127 , 255 , 122 , 48 , 6 , 0 , 0
5030 DATA  0 , 0 , 1 , 63 , 255 , 255 , 255 , 0 , 255 , 255 , 255 , 235 , 65 , 102 , 0 , 0 , 0 , 0 , 192 , 224 , 254 , 254 , 224 , 0 , 255 , 255 , 255 , 174 , 6 , 100 , 0 , 0
```

A linha 5 estabelece as cores da tela — **COLOR** — e reserva espaço na memória para acomodar o cordão **A$**, que é muito longo — **CLEAR 300**.

A linha 10 seleciona a tela de 32 colunas, com sprites grandes e em baixa resolução (32 por 32 pontos). Além disso, desativa a impressão das teclas de função na parte inferior da tela.

O **FOR...NEXT** das linhas 20 a 40 lê as linhas **DATA** do final do programa, onde está definido o padrão do tanque. Estes valores são transferidos para o cordão **A$** (veja página 188).

As linhas 50 a 100 criam os sprites que montam o tanque. Observe que o cordão **A$** é "recortado" pelas funções **MID$** e **LEFT$**. Os sprites criados são: 0, parte posterior do tanque da direita da figura 2; 1, parte dianteira atirando; 2, parte dianteira do tanque da esquerda atirando; 3, parte traseira. Os sprites 4 e 5 contêm a parte dianteira do tanque, cada uma apontada para um lado, sem o tiro.

As linhas 110 a 112 desenham o tanque em sua posição inicial. As linhas 120 a 160 mudam o valor de **X** e **Y**, conforme a tecla do cursor pressionada, movimentando o tanque. As condições que se seguem à conjunção **AND** evitam que o tanque ultrapasse os limites da tela. **F** é um sinalizador da direção em que o tanque aponta.

A linha 200 utiliza **F** para decidir em que direção desenhar o tanque. O desenho propriamente dito é feito pelas sub-rotinas 500 e 600.

Note que os sprites da parte dianteira e traseira são desenhados em níveis de prioridade diferentes. Se ambos estivessem no mesmo nível, o segundo a ser desenhado faria o primeiro desaparecer. Esta mesma propriedade responde pelo desaparecimento do tanque de sua antiga posição, quando a nova é desenhada.

As próximas linhas farão o tanque atirar ao toque da barra de espaço.

```
170 IF ASC(K$)=32 THEN IF F=1 THEN F=3 ELSE F=4
200 ON F GOSUB 500,600,700,800
700 PUT SPRITE 0,(X,Y),12,0
710 PUT SPRITE 1,(X+32,Y),12,1
720 FOR I=1 TO 50:NEXT
730 F=F-2:GOSUB 500:RETURN
800 PUT SPRITE 0,(X-16,Y),12,2
810 PUT SPRITE 1,(X+16,Y),12,3
820 FOR I=1 TO 50:NEXT
830 F=F-2:GOSUB 600:RETURN
```

Para produzir o tiro, o programa usa o sinalizador F. Conforme se tenha pressionado a barra de espaço, também são utilizados os sprites 1 e 2, correspondentes à parte dianteira dando tiro — nos dois sentidos.

Se você quiser sprites grandes em alta resolução, faça as seguintes modificações:

```
10 SCREEN 1,2:KEY OFF
111 PUT SPRITE 0,(X-8,Y),12,5
112 PUT SPRITE 1,(X+8,Y),12,3
500 PUT SPRITE 0,(X,Y),12,0
510 PUT SPRITE 1,(X+16,Y),12,4
600 PUT SPRITE 0,(X-8,Y),12,5
610 PUT SPRITE 1,(X+8,Y),12,3
700 PUT SPRITE 0,(X,Y),12,0
710 PUT SPRITE 1,(X+16,Y),12,1
800 PUT SPRITE 0,(X-8,Y),12,2
810 PUT SPRITE 1,(X+8,Y),12,3
```

UM SAPO

O programa a seguir cria o sapo da figura 2 e o faz pular ao toque da barra de espaço.

```
5 CLEAR 300:COLOR 12,15,12
10 SCREEN 1,3:KEY OFF
20 FOR I=1 TO 12*8
30 READ A:A$=A$+CHR$(A)
40 NEXT I
50 SPRITE$(0)=LEFT$(A$,32)
60 SPRITE$(1)=MID$(A$,33,32)
70 SPRITE$(2)=MID$(A$,65,32)
110 X=50:Y=120
111 PUT SPRITE 0,(X,Y),12,0
120 K$=INKEY$:IF K$="" THEN 120
170 IF ASC(K$)=32 THEN GOSUB 500
210 GOTO 120
500 PUT SPRITE 0,(X+32,Y-64),12,1
510 PUT SPRITE 1,(X+16,Y-32),12,2
520 FOR I=1 TO 100:NEXT
530 PUT SPRITE 0,(X+64,Y-64),12,1
540 PUT SPRITE 1,(X+48,Y-32),12,2
550 FOR I=1 TO 100:NEXT
560 PUT SPRITE 0,(X+96,Y),12,0
570 PUT SPRITE 1,(X+48,209),12,[illegible]
580 X=X+96
590 RETURN
5000 DATA  0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 4 , 15 , 31 , 63 , 127 , 254 , 248 , 127 , 0 , 0 , 0 , 0 , 0 , 128 , 192 , 176 , 96 , 240 , 224 , 192 , 192 , 32 , 28 , 0
5010 DATA  0 , 0 , 0 , 0 , 0 , 1 , 3 , 7 , 7 , 7 , 15 , 30 , 54 , 38 , 70 , 70 , 8 , 28 , 27 , 70 , 255 , 254 , 252 , 249 , 250 , 196 , 0 , 0 , 0 , 0 , 0 , 0
5020 DATA  0 , 0 , 0 , 1 , 3 , 6 , 2 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 140 , 144 , 144 , 32 , 32 , 48 , 32 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
```

A preparação do computador e a criação dos sprites a partir das linhas **DATA** são muito parecidas com o programa do tanque, só que o laço **FOR...NEXT** é mais curto.

A diferença é que não precisamos mover o sapo, apenas fazê-lo saltar quando a tecla de espaço for pressionada. As linhas 120 e 170 detectam a ocorrência de pressão na tecla.

A sub-rotina 500 é responsável pela trajetória que o sapo descreve. Para entender como os sprites foram montados, use **PUT SPRITE** no modo imediato, após ter parado o programa com **<CNTRL> <STOP>**.

Para obter o mesmo efeito com sprites de alta resolução, faça as seguintes modificações:

```
10 SCREEN 1,2:KEY OFF
500 PUT SPRITE 0,(X+16,Y-32),12,1
510 PUT SPRITE 1,(X+8,Y-16),12,[illegible]
530 PUT SPRITE 0,(X+32,Y-32),12,1
540 PUT SPRITE 1,(X+24,Y-16),12,2
560 PUT SPRITE 0,(X+48,Y),12,0
570 PUT SPRITE 1,(X+48,209),12,2
580 X=X+48
```

PROGRAMAÇÃO PARA JOYSTICKS

Os joysticks tornam seus jogos muito mais profissionais. E não se preocupe: você não precisa saber linguagem de máquina para utilizá-lo — o BASIC é suficiente.

Os jogos vendidos no comércio e os produzidos em casa apresentam, em geral, uma diferença evidente: os primeiros oferecem ao usuário a opção de usar um joystick; os segundos, não. Todavia, não é necessário mergulhar nas profundezas da programação em código para incluir este periférico nos programas de jogos.

Você verá neste artigo como utilizar um joystick em programas BASIC, tornando seus jogos mais profissionais e, sobretudo, muito mais divertidos.

Antes de fazer um programa que o inclua, é preciso obter um joystick compatível com o micro em questão. Os programas que aqui apresentamos devem ser utilizados com o joystick padrão disponível para cada computador. Observações a respeito se encontram antes de cada programa. Mais informações sobre joysticks em geral podem ser obtidas no artigo da página 287.

Existem vários tipos de joystick compatíveis com o Spectrum. Não é possível escrever um programa que funcione com todos eles, pois cada um tem sua própria maneira de se comunicar com a máquina. Assim, optamos por um programa que funcione com o tipo mais comum de joystick: Atari.

UMA ALÇA DE MIRA ANIMADA

A primeira parte do programa permite a movimentação de uma alça de mira na tela do Spectrum.

```
100 BORDER 1: PAPER 1: INK 7:
OVER 1: CLS : INK 8
110 FOR n=USR "a" TO USR "h"+7
: READ a: POKE n,a: NEXT n
130 LET s=0: LET x=15: LET y=
10
140 PRINT  OVER 1;AT y,x;CHR$
148;CHR$ 149;AT y+1,x;CHR$ 150
;CHR$ 151
200 GOSUB 500
480 GOTO 200
500 LET i$=INKEY$: IF i$=""
THEN  RETURN
505 LET i=CODE i$
510 PRINT  OVER 1;AT y,x;CHR$
```

- COMPATIBILIDADE DOS DIFERENTES TIPOS DE JOYSTICKS
- COMO UTILIZAR JOYSTICKS EM PROGRAMAS BASIC
- UMA ALÇA DE MIRA ANIMADA
- TORNE SEUS JOGOS MAIS PROFISSIONAIS
- UM TESTE PARA SEU JOYSTICK

```
 148;CHR$ 149;AT y+1,x;CHR$ 150
 ;CHR$ 151
 520 IF i=57 AND y>1 THEN  LET
 y=y-1
 530 IF i=56 AND y<20 THEN  LET
 y=y+1
 540 IF i=55 AND x<30 THEN  LET
 x=x+1
 550 IF i=54 AND x>0 THEN  LET
 x=x-1
 560 PRINT  OVER 1;AT y,x;CHR$
 148;CHR$ 149;AT y+1,x;CHR$ 150
 ;CHR$ 151
 570 RETURN
1000 DATA 14,27,127,31,15,7,15,
31
1010 DATA 0,0,0,0,0,192,112,188
1020 DATA 31,29,30,15,3,1,1,3
1030 DATA 206,30,124,248,224,64
,64,224
1040 DATA 12,12,12,12,252,252,0
,0
1050 DATA 48,48,48,48,63,63,0,0
1060 DATA 0,0,252,252,12,12,12,
12
1070 DATA 0,0,63,63,48,48,48,48
```

O programa começa estabelecendo as cores do vídeo. Em seguida, utilizando os valores das linhas **DATA** 1000 a 1070, cria oito caracteres definidos pelo usuário. O programa não usa todos os caracteres, pois eles desenham não só a alça de mira, mas também um pato. Este será utilizado pela parte restante do programa, que apresentaremos num próximo artigo.

Criados os caracteres, a linha 130 acerta a posição inicial da mira na tela. Além disso, zera o contador de pontos — ou escore —, que também aguarda a próxima parte do programa para ser utilizado.

A linha 140 usa o comando **PRINT** para colocar na tela a metade superior da mira, usando dois caracteres definidos pelo usuário. Depois, coloca a metade inferior, usando dois outros caracteres. A alça de mira é controlada pela sub-rotina que começa na linha 500, chamada linha 200.

A sub-rotina que controla o joystick usa a função **INKEY$** para saber em que direção o bastão está virado. Isto é possível porque o joystick envia caracteres ao micro, como se fizesse parte do teclado. Os códigos desses caracteres estão na figura 2. As quatro direções principais dão os valores 57, 56, 55 e 54.

1. A função *STICK(n)* do MSX assume valores diferentes conforme a direção dada pelo bastão do joystick.

2. O joystick do Atari envia caracteres ao Spectrum. Os códigos mudam conforme a direção.

3. Estes são os códigos dos caracteres enviados ao computador pelo joystick do TK-85.

A primeira linha da sub-rotina — linha 500 — verifica se o joystick está na posição central. Nenhum caractere será enviado se o bastão ocupar esta posição. Em seguida, a linha 505 guarda o código do caractere na variável i, usando a função **CODE**.

A linha 510 apaga a mira, pois é a segunda vez que **PRINT OVER 1** foi usado (a primeira foi na linha 140). **OVER 1** faz com que o gráfico impresso na primeira vez desapareça quando é usado pela segunda vez.

Agora que a posição antiga foi apagada, a nova pode ser calculada. Ela vai depender da direção em que o bastão do joystick for empurrado pelo jogador. A linha 520 percebe o movimento para cima; a linha 530 percebe os movimentos para baixo; e as linhas 540 e 550 percebem os movimentos laterais.

A sub-rotina termina colocando a alça de mira em sua nova posição. Observe que, como é a primeira vez que a figura é desenhada nesta posição, ela aparece normalmente.

Para permitir um movimento contínuo da mira, a linha 480 traz um **GOTO 200**, fazendo com que a sub-rotina do joystick seja chamada repetidamente.

Como o ZX-81 não permite a movimentação de figuras usando só o BASIC, a mira desenhada pelo programa a seguir é apenas um ponto gráfico (um quarto de caractere).

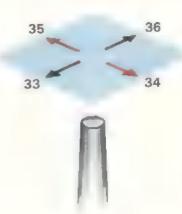
O programa deve ser utilizado com o joystick do TK-85 da Microdigital.

```
10 LET X=32
20 LET Y=20
30 GOTO 170
40 LET LX=X
50 LET LY=Y
100 LET A$=INKEY$
110 IF A$="" THEN GOTO 100
115 LET A=CODE A$
120 IF A=33      X>0 THEN LET X=
X-1
130 IF A=36 AND X<63 THEN LET X
=X+1
140 IF A=34 AND Y>0 THEN LET Y=
Y-1
150 IF A=35 AND Y<40 THEN LET Y
=Y+1
160 UNPLOT LX,LY
170 PLOT X,Y
180 GOTO 40
```

As linhas 10 e 20 armazenam a posição central da tela gráfica em **X** e **Y**. A linha 30 desvia o programa para a linha 170, para que o primeiro ponto seja impresso. Se ela for suprimida, o primeiro ponto só aparecerá após algum movimento do joystick.

As linhas 40 e 50 guardam os últimos valores assumidos por **X** e **Y** em **LX** e **LY**, para que o programa saiba onde apagar a última posição do ponto.

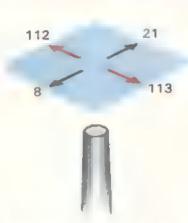
As linhas de 100 a 150 controlam o joystick. A linha 100 promove uma "varredura" do teclado. Isto é necessário porque o joystick envia caracteres ao micro, como se fizesse parte do teclado. A linha 110 repete a linha 100 até que algum movimento seja feito no joystick. Ao ocorrer o movimento, a linha 115 guarda o código do caractere enviado na variável **A**, usando **CODE**.

Os códigos dos caracteres enviados pelo joystick são 35 (para cima), 34 (para baixo), 33 (esquerda) e 36 (direita) (veja a figura 3). As linhas 150, 140, 120 e 130 detectam, respectivamente, estes movimentos. As condições sobre **X** e **Y**, que se seguem à conjunção **AND** nestas mesmas linhas, evitam que o ponto ultrapasse os limites da tela.

Finalmente, a linha 160 apaga o ponto de sua última posição, usando **UNPLOT**, e a linha 170 coloca-o em sua nova posição com **PLOT**. A linha 180 volta ao princípio.

O MSX tem uma função especial — **STICK (N)** —, que facilita muito o controle de joysticks por programas em BASIC. Além disso, essa função permite que nosso programa seja utilizado mesmo por quem não possui um joystick.

COMO MOVER SPRITES COM O [illegible]

O programa a seguir foi feito para funcionar com o joystick do HOTBIT conectado à tomada frontal direita.

```
10        1,2:KEY OFF
20 FOR I=1 TO 32
30 READ A:A$=A$+CHR$(A)
40 NEXT I
50 SPRITE$(0)=A$
60 PUT SPRITE 0,(115,85),15
170 X=115:Y=85
200 GOSUB 1000
210 GOTO 200
1000 A=STICK(1)
1010 ON A GOTO 1030,1040,1050,1
060,1070,1080,1090,1100
1020 RETURN
1030 Y=Y-1:GOTO 1110
1040 X=X+1:Y=Y-1:GOTO 1110
1050 X=X+1:GOTO 1110
1060 X=X+1:Y=Y+1:GOTO 1110
1070 Y=Y+1:GOTO 1110
1080 X=X-1:Y=Y+1:GOTO 1110
```

4. Cada direção corresponde a um caractere no joystick do TK-2000. Estes são seus códigos.

```
1090 X=X-1:GOTO 1110
1100 X=X-1:Y=Y-1:GOTO 1110
1110 PUT SPRITE 0,(X,Y),15
1120 RETURN
5000 DATA  3 , 12 , 16 , 32 , 3
2 , 64 , 64 , 127 , 64 , 64 , 3
2 , 32 , 16 , 12 , 3 , 0 , 224
, 152 , 132 , 130 , 130 , 129 ,
 129 , 255 , 129 , 129 , 130 ,
130 , 132 , 152 , 224 , 0
```

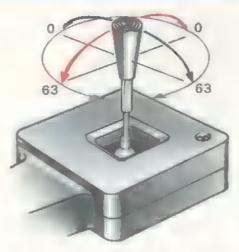
A linha 10 ativa a tela de 32 colunas, com sprites grandes em alta resolução — **SCREEN 1,2**. Além disso, impede que o conteúdo das teclas de função seja impresso na parte inferior da tela, usando **KEY OFF**.

As linhas 20 a 50 criam um sprite grande — dezesseis por dezesseis — para a alça de mira. Os números que definem o padrão do sprite são lidos na linha 5000 (veja página 188). A mira é então desenhada em sua posição inicial pela linha 60. A linha 170 guarda esta posição em **X** e **Y**.

A linha 200 chama a sub-rotina de controle do joystick, que movimentará o sprite uma vez. A linha 210 faz com que o processo se repita, possibilitando o movimento contínuo.

As linhas 1000 a 1120 controlam o joystick, por meio da função especial **STICK (1)**. Esta assume diferentes valores, conforme a posição do joystick número 1 (veja a figura 1). **STICK (2)** é usada com o joystick 2; **STICK (0)** possibilita a substituição do joystick pelas teclas do cursor.

A linha 1000 armazena o valor correspondente à posição do joystick em **A**. A linha 1010 usa **ON A GOTO** para desviar o programa conforme o valor de **A** (veja página 76). Isto pode ser feito porque os números que o joystick envia são: 0, em repouso; 1, norte; 2, nordeste; 3, leste; 4, sudeste; 5, sul; 6, sudoeste; 7, oeste e 8, noroeste. As linhas para onde o programa é desviado calculam os novos valores de **X** e **Y** — linhas 1030 a 1100. Quando **A** é igual a zero, o programa vai para a linha 1020.

Depois que a nova posição foi calculada, a linha 1110 desenha o sprite nela. A linha 1120 provoca o retorno da sub-rotina.

5. O joystick do TRS-Color é analógico; cada um dos seus dois potenciômetros fornece um valor entre 0 e 63.

6. A leitura do joystick analógico do Apple só pode ser feita por um programa em linguagem de máquina.

Por meio do comando **PDL** (*paddle* — em inglês, raquete) controla-se adequadamente o joystick do Apple. Este micro possui uma entrada de joystick que, na verdade, é uma tomada, onde quase todo tipo de aparelho analógico ou digital pode ser conectado sem maiores dificuldades.

O programa a seguir foi feito totalmente em BASIC; por isso, não controla muito bem o joystick analógico.

DESCUBRA OS CÓDIGOS

Se seu joystick é de outra marca, experimente este pequeno programa para descobrir quais são os códigos dos caracteres enviados por ele.

Caso você tenha um Apple, não utilize nenhum programa. Pode ser que seu joystick utilize outro byte analógico. Existem quatro desses bytes, endereços −16284 a −16281. Tente todas as combinações possíveis nas linhas 510 e 520.

```
10 LET A$ = INKEY$
20 IF A$ = "" THEN GOTO 10
30 PRINT CODE A$
40 GOTO 10
```

Conecte o joystick na tomada direita do HOTBIT. Se não funcionar no seu MSX, troque a tomada ou substitua o número 1 por 2 na função **STICK**:

```
10 PRINTSTICK(1);:GOTO 10
```

```
10 GET A$
20 PRINT ASC(A$)
30 GOTO 10
```

```
10  HOME :E = 35000: HIMEM: E
20 F =  INT (E / 256): POKE 232
,E - F * 256: POKE 233,F
30  FOR I = E TO E + 41 + 5 * 3
2
40  READ A: POKE I,A
50  NEXT
60  HGR : SCALE= 1: ROT= 0:X =
130:Y = 90
200  GOSUB 500
210  GOTO 200
500 LX = X:LY = Y
510 A =  PDL (1):B =  PDL (0)
530  IF A = 0 AND Y > 8 THEN Y
= Y - 8: GOTO 580
540  IF A = 255 AND Y < 144 THE
N Y = Y + 8: GOTO 580
550  IF B = 0 AND X > 8 THEN X
= X - 8: GOTO 580
560  IF B = 255 AND X < 266 THE
N X = X + 8: GOTO 580
580  HCOLOR= 0
590  DRAW 5 AT LX,LY
600  HCOLOR= 3
610  DRAW 5 AT X,Y
```

Podemos usar joysticks [illegible] [illegible] programas de jogos [illegible] publicados [illegible] INPUT?

A parte principal do programa fornecido neste artigo pode [illegible] adicionada [illegible] [illegible] dos programas que [illegible] **GET$** [illegible] **INKEY$** para "ler" o teclado. Assim, você poderá aperfeiçoar seus jogos, pois é muito [illegible] conveniente e divertido [illegible] joysticks do que teclado.

As linhas importantes destes programas são: para [illegible] Spectrum, linhas 520 a 550; para o ZX81, linhas 100 a 150; para o MSX, linhas 1000 [illegible] 1120; para o TK-2000 [illegible] Apple, linhas 500 a 530; e para o TRS-Color, as linhas 1000 a 1070, chamadas [illegible] sub-rotina (alguns valores terão que ser modificados, conforme [illegible] tamanho da figura).

```
620  RETURN
2000  DATA  20 ,0 ,42 ,0 ,74 ,0
 ,106 ,0 ,138 ,0 ,170 ,0 ,202 ,
▪ ,234 ,0 ,10 ,1 ,42 ,1 ,74 ,1
,106 ,1 ,138 ,1 ,170 ,1 ,202 ,1
 ,234 ,1 ,10 ,2 ,42 ,2 ,74 ,2 ,
106 ,2 ,138 ,2
2010  DATA   0 ,72 ,9 ,45 ,141
,59 ,31 ,255 ,83 ,45 ,45 ,45 ,2
13 ,63 ,63 ,223 ,74 ,9 ,45 ,173
 ,59 ,255 ,219 ,74 ,9 ,45 ,173
,59 ,63 ,255 ,19 ,0
2020  DATA   0 ,72 ,41 ,45 ,173
 ,251 ,63 ,223 ,74 ,41 ,45 ,141
 ,59 ,63 ,223 ,83 ,73 ,73 ,213
,219 ,219 ,83 ,73 ,73 ,209 ,255
 ,219 ,19 ,0 ,0 ,0 ,0
2030  DATA   0 ,72 ,73 ,73 ,218
 ,219 ,219 ,74 ,73 ,73 ,218 ,21
9 ,219 ,74 ,73 ,73 ,218 ,219 ,2
7 ,119 ,45 ,77 ,137 ,219 ,63 ,2
55 ,6 ,0 ,0 ,0 ,0 ,0
2040  DATA   0 ,40 ,77 ,45 ,141
 ,27 ,63 ,255 ,83 ,45 ,45 ,77 ,
218 ,27 ,63 ,63 ,46 ,109 ,73 ,2
09 ,219 ,27 ,31 ,110 ,77 ,73 ,2
18 ,219 ,63 ,55 ,0 ,0
2050  DATA   0 ,72 ,73 ,73 ,218
 ,27 ,223 ,83 ,73 ,77 ,209 ,219
 ,223 ,83 ,45 ,45 ,45 ,213 ,219
 ,223 ,83 ,73 ,77 ,209 ,219 ,22
3 ,19 ,0 ,0 ,0 ,0 ,0
```

As linhas 10 a 50 criam uma tabela de figuras a partir das linhas **DATA** calculadas com o editor da página 316.

A linha 60 estabelece as condições iniciais de posição, cor e escala na tela gráfica.

A sub-rotina 500 controla o joystick. A linha 210 chama repetidamente [illegible] sub-rotina, possibilitando um movimento contínuo.

As linhas 510 e 520 lêem o status dos dois *paddles* conectados ao joystick, por meio dos comandos **PDL(0)** e **PDL(1)**.

As linhas 530 a 560 deslocam a mira continuamente para a esquerda e para cima, a menos que um movimento para baixo ou para a direita seja executado. Isto [illegible] o máximo que se pode fazer utilizando apenas o BASIC.

As condições que se seguem às conjunções **AND** evitam que a mira ultrapasse os limites da tela.

As linhas 580 e 590 apagam a mira de sua última posição; as linhas 600 e 610 refazem o desenho na nova.

A tabela de figuras criada a partir das linhas **DATA** inclui um pato, além da mira. Ele será utilizado num próximo artigo.

O programa anterior não funcionará no TK-2000. Faça as modificações indicadas a seguir e use o joystick da Microdigital ou as setas do teclado.

```
510  GET A$
520  A =  ASC (A$)
530  IF A = 112 AND Y > 8 THEN
Y = Y - 8: GOTO 570
540  IF A = 113 ▪▪▪ Y < 144 THE
N Y = Y + 8: GOTO 570
550  IF ▪ = 8 AND X > ▪ THEN X
= X - 8: GOTO 570
560  IF A = 21 AND X < 266 THEN
 ▪ = X + 8: GOTO 570
570  REM
```

O joystick envia caracteres ao micro, como se fizesse parte de seu teclado: assim, a linha 510 utiliza **GET$ A$** para obter a nova posição do joystick. A linha 120 guarda o código do caractere enviado em **A**.

Os códigos correspondentes aos movimentos do joystick estão na figura 4. Esses códigos são os mesmos das setas do teclado, motivo pelo qual não é necessário dispor de um joystick para usar o programa.

A linha 530 detecta o movimento para cima; a 540, para baixo; a 550 para a esquerda [illegible] [illegible] 560 para a direita. As condições que se seguem [illegible] conjunção **AND** nestas linhas evitam que a mira ultrapasse [illegible] limites da tela.

As operações após **THEN** calculam as novas coordenadas.

Note que o joystick do TK-2000 não é auto-repetitivo. A tecla **REPEAT** contorna parcialmente o problema.

O TRS-Color tem uma função em BASIC para controlar o joystick — **JOYSTK** — que facilita muito emprego deste periférico. O computador admite a utilização de dois joysticks, mas o programa abaixo só usa um.

Antes de passar ao programa, conecte seu joystick na tomada traseira marcada com **JOY-DIR**. O programa não funcionará se o drive de disquetes estiver conectado.

UMA ALÇA DE MIRA ANIMADA

Digite a primeira seção do programa [illegible] depois rode-a. Você verá uma alça de mira surgir na tela.

```
10 PMODE 3,1
20 FOR K=1536 TO 1868 STEP 32
30 FOR J=0 TO 2
40 READ A:POKE K+J,A
50 NEXT J,K
160 SCREEN 1,0
170 GOTO 170
4000 DATA 252,15,192,192,0,192,
48,3,0,12,12,0,3,48,0
4010 DATA 0,0,0,3,48,0,12,12,0,
48,3,0,192,0,192,251,15,192
```

Esta parte do programa é bem simples. As linhas 20 e 50 usam **POKE** para colocar a mira na tela, com os valores dados pelas linhas **DATA** 4000 e 4010.

A linha 160 ativa [illegible] tela de alta resolução. A linha 170 é temporária e serve para manter a tela gráfica ligada.

Depois de digitar a segunda parte do programa, você poderá movimentar a mira pela tela usando o joystick.

```
60 DIM S(5),B(5),D(4),H(4)
70 GET (0,0)-(17,11),S,G
130 PCLS
140 LINE(0,0)-(255,191),PSET,B
170 X=127:Y=95
200 GOSUB 1000
210 GOTO 200
1000 J0=JOYSTK(0):J1=JOYSTK(1)
1010 IF J0>58 THEN J0=58
1020 IF J1>59 THEN J1=59
1030 IF X=J0*4+10 ▪▪▪ Y=J1*3+6
THEN 1070
1040 PUT(X-8,Y-5)-(X+9,Y+5),B,P
SET
1050 X=J0*4+10:Y=J1*3+6
1060 PUT (X-8,Y-5)-(X+9,Y+5),S,
OR
1070 RETURN
```

Embora o programa empregue apenas duas matrizes para armazenar desenhos, a linha 60 dimensiona quatro delas — duas para uso futuro. A linha 70, com **GET**, armazena o desenho da mira na matriz **S**. Esta matriz é usada juntamente com a matriz [illegible] — em branco

Dependendo do computador, pode-se criar uma alça de mira animada a partir de caracteres definidos pelo usuário, dos caracteres gráficos da ROM ou mesmo do conjunto de caracteres padrão.

— para animar o desenho.

A linha 130 limpa a tela para que, quando ela for ligada, o desenho original da mira não apareça. A linha 140 desenha uma moldura para tornar o jogo mais bonito.

Antes da sub-rotina de controle do joystick ser chamada pelo **GOSUB** da linha 200, a posição inicial da mira é acertada pelos valores de **X** e **Y** da linha 170.

Trataremos agora da parte mais importante do programa: a sub-rotina que faz com que a posição da mira na tela corresponda à posição do joystick. Ela vai da linha 1000 à linha 1070.

A linha 1000 usa a função **JOYSTK. JOYSTK (0)** verifica a posição horizontal do **joystick** direito, enquanto **JOYSTK (1)** verifica a posição vertical do mesmo joystick. **JOYSTK (2)** e **JOYSTK (3)** fazem o mesmo para o joystick esquerdo.

Cada uma destas quatro funções pode assumir um valor que vai de 0 a 63, de acordo com a posição dos joysticks.

Na sub-rotina, a linha 1000 usa as variáveis **J1** e **J0** para armazenar o valor de **JOYSTK (1)** e **JOYSTK (0)**: assim, não é preciso escrever o nome todo da função ao longo do programa. Este procedimento é bem comum. Em INPUT, utilizamos **K$** para armazenar o valor de **INKEY$**, por exemplo.

As linhas 1010 a 1020 impedem que a mira saia fora dos limites da tela, levando em conta o tamanho do desenho.

As linhas 1040 a 1060 são responsáveis pela animação. Primeiro, apagam a mira da sua última posição; depois, calculam a nova posição e nela situam a mira, usando **PUT**.

A linha 1040 apaga o desenho em sua última posição colocando a matriz em branco **B**, com **PUT**. A nova posição é calculada a partir de **J1** e **J0** (veja a linha 1050). **X** e **Y** são as novas coordenadas do centro da mira. Observe que, enquanto **J0** é multiplicado por 4, **J1** é multiplicado por 3. Esses fatores de multiplicação são determinados pela resolução da tela que se utilizou. No nosso caso, ela é de 0 a 255 pontos na horizontal, e de 0 a 190 na vertical. Quando multiplicamos os valores dos **JOYSTK**, estamos adaptando o intervalo de 0 a 63 do joystick aos intervalos de 0 a 255 e 0 a 191 da tela. Esses valores são os mesmos para qualquer **PMODE**. Você só precisará alterá-los se quiser considerar o tamanho de outra figura que estiver utilizando. O uso de fatores menores deixa mais espaço para movimentar figuras grandes.

Calculada a nova posição da mira, a linha 1060 coloca o desenho na tela, usando **PUT...OR** para não prejudicar o que estiver desenhado naquelas posições.

A sub-rotina funcionará como já foi explicado após a adição do **RETURN** da linha 1070. Se o joystick não mudasse de posições, a mira ficaria piscando na tela, uma vez que estaria sendo constantemente apagada e desenhada de novo. Para evitar que isso ocorra, a linha 1030 verifica se a posição mudou. Caso o joystick não tenha se movido, o programa continua na linha 1070, evitando as linhas de animação.

MAIS REQUINTE EM SEUS DESENHOS

Já aprendemos a ■ funções matemáticas para fazer gráficos circulares. Veremos agora como empregar **SIN** ■ **COS** ■ elaboração ■ desenhos mais sofisticados.

Em artigo anterior, na página 334, vimos algumas das funções matemáticas existentes ■ computadores, bem como sua utilidade enquanto ferramentas gráficas. Agora, examinaremos mais de perto sua atuação e indicaremos outros usos para elas.

As funções que nos interessam no momento são o ■ e o cosseno; caso ainda não esteja familiarizado com elas, aconselhamos consultar o artigo citado. Nele, você verá como estas funções estão relacionadas com a posição de um ponto ■ redor de um círculo, e como usá-las para calcular as coordenadas de qualquer ponto. O programa da bússola, no artigo da página 334, utiliza o seno ■ o cosseno para posicionar as marcações de número em seus devidos lugares, ao redor da bússola.

COMO DESENHAR UM RELÓGIO

Os programas apresentados até agora não têm muita utilidade prática. Porém, como veremos detalhadamente nos próximos artigos, as funções trigonométricas oferecem variadas possibilidades de uso. Por enquanto, vamos nos concentrar no desenho de ■ relógio. Você já deve ter pelo menos uma idéia de como fazê-lo, com base no primeiro artigo da série.

O princípio é ■ mesmo que orientou ■ desenho da bússola, com uma diferença: ■ vez de fornecermos um ângulo, o computador precisará selecionar sozinho os ângulos a serem mostrados, aumentando-os com o tempo.

Os programas ■ seguir calculam ■ posições dos ponteiros, usando **SIN** e **COS**, para que mostrem a hora certa.

Embora seja fácil montar um relógio de doze horas, este programa desenha um relógio de 24 horas, com numeração de 1 ■ 24 em seu mostrador. Dessa maneira, poderemos examinar melhor as rotinas que imprimem ■ números. Depois de digitar ■ rodar ■ programa, veremos o que **SIN** e **COS** fazem exatamente.

No programa da bússola, os números ao redor do aparelho marcavam os vários graus; no relógio, marcam as horas. Os números de 1 a 24 são selecio-

- FAÇA UM RELÓGIO MECÂNICO
- COMO POSICIONAR OS NÚMEROS
- DESENHE OS PONTEIROS
- ACRESCENTE UM ALARME
- COMO ADICIONAR NÚMEROS ROMANOS
- DESENHOS ABSTRATOS
- O USO DE **SIN** E **COS** EM DESENHOS MAIS ELABORADOS

nados por um laço **FOR...NEXT** e impressos na tela em coordenadas determinadas por **SIN** e **COS**.

Parte do programa para o TRS-Color e [illegible] MSX parecerão familiares a quem usou a rotina para desenhar caracteres na tela de alta resolução, dada no artigo da página 232. Repetimos esta rotina no programa dado [illegible] seguir porque [illegible] computador não imprime caracteres na tela em **PMODE 4** ou **SCREEN 2** e 3, mas com ela podemos colocar os números ao redor do marcador do relógio.

Se você salvou [illegible] rotina citada, carregue-a no seu computador para não precisar digitá-la novamente. Depois, acrescente essas linhas [illegible] programa abaixo.

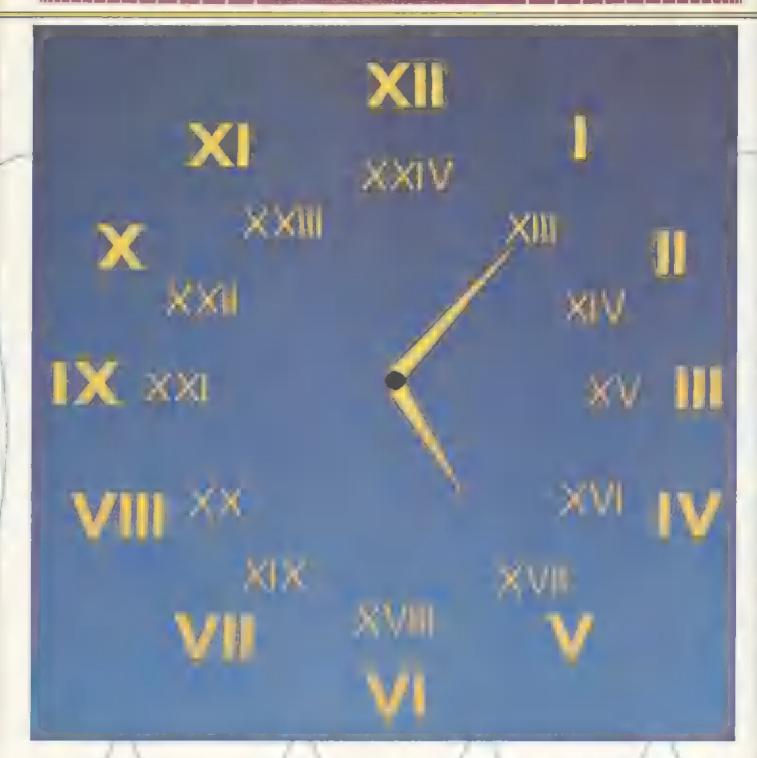
A mesma rotina também serve para colocar números no programa da bússola visto anteriormente; basta adaptá-lo para isso. Quando [illegible] marcador é desenhado, o computador automaticamente desenha o ponteiro de segundos do relógio; este funciona como o ponteiro de segundos de um relógio real. O ângulo inicial, no topo do círculo, é de 0 graus, [illegible] aumenta gradualmente até chegar nos 360 graus, novamente no topo. A velocidade do ponteiro de segundos é determinada por um comando **PAUSE** ou pelo temporizador no computador.

Os ponteiros do relógio são movimentados por meio de um laço **FOR...NEXT**, no Spectrum. O programa do TRS-Color usa uma variável que é atualizada pelo temporizador cada vez que [illegible] roda o programa (este é um laço contínuo). O ponteiro de segundos movimenta-se sempre que [illegible] variável acusa que já se passou um segundo.

S

```
 5 CIRCLE 134,92,70
 6 LET q=-1
10 FOR n=1 TO 24
20 PRINT AT 10-10*COS (n/12*
PI),16+10*SIN (n/12*PI);n
30 NEXT n
40 FOR t=0 TO 20000
50 LET a=t/30*PI
60 LET sx=65*SIN a: LET sy=65
*COS a
70 PLOT 134,92: [illegible]  OVER 1;
sx,sy
[illegible] PAUSE 42
 90 PLOT 134,92: DRAW  OVER 1;
sx,sy
100 NEXT t
```

T

```
10 PMODE 4,1
20 DIM LE$(26)
30 PCLS
40 FOR K=0 TO 26:READ LE$(K):NE
XT
50 FOR K=0 TO 9:READ NU$(K):NEX
T
60 DATA BR2,ND4R3D2NL3ND2BE2,ND
4R3DGNL2FDNL3BU4BR2,NR3D4R3BU4B
R2,ND4R2FD2GL2BE4BR,NR3D2NR2D2R
3BU4BR2
70 DATA NR3D2NR2D2BE4BR,NR3D4R3
U2LBE2BR,D4BR3U2NL3U2BR2,ND4BR2
,BD4REU3L2R3BR2,D2ND2NF2E2BR2
[illegible] DATA D4R3BU4BR2,ND4FREND4BR2
,ND4F3DU4BR2,NR3D4R3U4BR2,ND4R3
D2NL3BE2,NR3D4R3NHU4BR2
90 DATA ND4R3D2L2F2BU4BR2,[illegible]
U2L3U2R3BR2,RND4RBR2,D4R2U4BR2,
D3FEU3BR2,D4EFU4BR2
100 DATA DF2DBL2UE2UBR2,DFND2EU
BR2,R3G3DR3BU4BR2
110 DATA NR2D4R2U4BR2,BDEND4BR2
,R2D2L2D2R2BU4BR2,NR2BD2NR2BD2R
2U4BR2,D2R2D2U4BR2,NR2D2R2D2L2B
E4,D4R2U2L2BE2BR2,R2ND4BR2,NR2D
4R2U2NL2U2BR2,NR2D2R2D2U4BR2
200 MX=127:SX=127:MY=95:SY=95
210 SCREEN 1,1
220 CIRCLE (127,95),60,1
230 PI=ATN(1)/15
240 FOR K=5 TO 120 STEP [illegible]
250 LINE(127+55*SIN(K*PI),95-55
*COS(K*PI))-(127+59*SIN(K*PI),9
5-59*COS(K*PI)),PSET
260 A$=MID$(STR$(K/5),2):DRAW"B
M"+STR$(INT(123+68*SIN(K*PI)))+
","+STR$(INT(92-68*COS(K*PI)))+
"C5S6":GOSUB 1000
270 NEXT [illegible]
280 IF TIMER<50 THEN 280
290 TIMER=0:T=T+1
300 IF T=86400 THEN T=0
310 H=T/3600
320 M=T/60-INT(H)*60
330 S=T-INT(M)*60-INT(H)*3600
340 LX=SX:LY=SY
350 SX=127+45*SIN(S*PI*2):SY=95
-45*COS(S*PI*2)
360 LINE(127,95)-(SX,SY),PSET
370 LINE(127,95)-(LX,LY),PRESET
380 LINE(127,95)-(MX,MY),PRESET
390 MX=127+30*SIN(M*PI*2):MY=95
-30*COS(M*PI*2)
400 LINE(127,95)-(MX,MY),PSET
```

```
410 LINE(127,95)-(HX,HY),PRESET
420 HX=127+20*SIN(H*PI*5):HY=95
-20*COS(H*PI*5)
430 LINE(127,95)-(HX,HY),PSET
440 GOTO 280
1000 FOR M=1 TO LEN(A$)
1010 B$=MID$(A$,M,1)
1020 IF B$>="0" AND B$<="9" THE
N DRAW NU$(VAL(B$)):GOTO 1050
1030 IF B$="" THEN N=0 ELSE N=A
SC(B$)-64
1040 DRAW LE$(N)
1050 NEXT
1060 RETURN
```

```
20 DIM LE$(26)
30 CLS
40 ▬▬ K=0 TO 26:READ LE$(K):NE
XT
50 ▬▬ K=0 TO 9:READ NU$(K):NEX
T
60 DATA BR2,ND4R3D2NL3ND2BE2,ND
4R3DGNL2FDNL3BU4BR2,NR3D4R3BU4B
R2,ND4R2FD2GL2BE4BR,NR3D2NR2D2R
3BU4BR2
70 DATA NR3D2NR2D2BE4BR,NR3D4R3
U2LBE2BR,D4BR3U2NL3U2BR2,ND4BR2
,BD4REU3L2R3BR2,D2ND2NF2E2BR2
80 DATA D4R3BU4BR2,ND4FREND4BR2
,ND4F3DU4BR2,NR3D4R3U4BR2,ND4R3
D2NL3BE2,NR3D4R3NHU4BR2
90 DATA ND4R3D2L2F2BU4BR2,BD4R3
U2L3U2R3BR2,RND4RBR2,D4R2U4BR2,
```

```
D3FEU3BR2,D4EFU4BR2
100 DATA DF2DBL2UE2UBR2,DFND2EU
BR2,R3G3DR3BU4BR2
110 DATA NR2D4R2U4BR2,BDEND4BR2
,R2D2L2D2R2BU4BR2,NR2BD2NR2BD2R
2U4BR2,D2R2D2U4BR2,NR2D2R2D2L2B
E4,D4R2U2L2BE2BR2,R2ND4BR2,NR2D
4R2U2NL2U2BR2,NR2D2R2D2U4BR2
200 MX=127:SX=127:MY=95:SY=95
210 SCREEN2
220 CIRCLE(127,95),60,1,,,1
230 PI=ATN(1)/15
240 FOR K=5 TO 120 STEP5
250 LINE(127+55*SIN(K*PI),95-55
*COS(K*PI))-(127+59*SIN(K*PI),9
5-59*COS(K*PI)),1
260 A$=MID$(STR$(K/5),2):DRAW"B
M"+STR$(INT(123+68*SIN(K*PI)))+
","+STR$(INT(92-68*COS(K*PI)))+
"C1S6":GOSUB 1000
270 NEXT K
280 IF TIME<50 THEN 280
290 TIME=0:T=T+1
300 IF T=86400! THEN T=0
310 H=T/3600
320 M=T/60-INT(H)*60
330 S=T-INT(M)*60-INT(H)*3600
340 LX=SX:LY=SY
350 SX=127+45*SIN(S*PI*2):SY=95
-45*COS(S*PI*2)
360 LINE(127,95)-(SX,SY),1
370 LINE(127,95)-(LX,LY),4
380 LINE(127,95)-(MX,MY),4
390 MX=127+30*SIN(M*PI*2):MY=95
-30*COS(M*PI*2)
400 LINE(127,95)-(MX,MY),1
410 LINE(127,95)-(HX,HY),4
420 HX=127+20*SIN(H*PI*5):HY=95
-20*COS(H*PI*5)
430 LINE(127,95)-(HX,HY),1
440 GOTO 280
1000 FOR M=1 TO LEN(A$)
1010 B$=MID$(A$,M,1)
1020 IFB$>="0" AND B$<="9" THEN
 DRAW NU$(VAL(B$)):GOTO 1050
1030 IFB$="" THEN N=0 ELSE N=AS
C(B$)-64
1040 DRAW LE$(N)
1050 NEXT
1060 RETURN
```

O ZX-81 não possui o comando **DRAW**; assim, para fazer o relógio, precisaríamos desenhar os ponteiros ponto por ponto, via comando **PLOT**. Além disso, sua tela gráfica não é de alta resolução, como nos outros computadores. Por ambos os motivos, uma versão para ele não ficaria muito boa; portanto, não a fizemos.

O capítulo 19 do manual do ZX-81 contém um programa de relógio que poderíamos tomar como exemplo, mas este relógio não tem ponteiros. Ao contrário dos que aqui apresentamos, é formado apenas por números que marcam a hora, sem o circulo como corpo.

O Apple também não possui o comando **DRAW**; por isso, uma rotina para desenhar letras e números em tela de alta resolução ficaria extremamente complicada.

Cada versão emprega uma rotina parecida para imprimir os números em suas posições corretas ao redor do marcador do relógio. Esta rotina mostra claramente como **SIN** e **COS** são usados para controlar as operações de impressão na tela. Ela se resume mais ou menos ao seguinte:

```
10 FOR n=1 TO 24
20 PRINT AT(coordenada x do
   centro da tela + 10*SIN(2*
   PI/n) , coordenada y do cen-
   tro da tela - 10*COS(2*PI/
   n));n
30 NEXT n
```

A maneira de manipular esta rotina varia de computador para computador, o que a faz parecer mais complicada do que realmente é. Determinada máquina pode, por exemplo, ajustar uma variável extra para o ângulo, em vez de usar (2*PI/n) no **PRINT AT**.

A rotina do Spectrum segue a mesma fórmula das outras, mas parece diferente, porque utiliza um método distinto para ■ **PRINT AT** numa posição da tela: em vez do primeiro número depois do comando ser a coordenada x, é a coordenada y. Além disso, o valor 0 para a coordenada y não corresponde à parte inferior da tela, mas ao topo desta.

Tente isto, no Spectrum:

```
PRINT AT 0,0;"a"
```

O "a" deve aparecer no canto superior esquerdo da tela.

Devido a esta peculiaridade do **PRINT AT** do Spectrum, nele o **COS** é a primeira função da linha, já que determina a posição vertical do ponto ao redor do círculo.

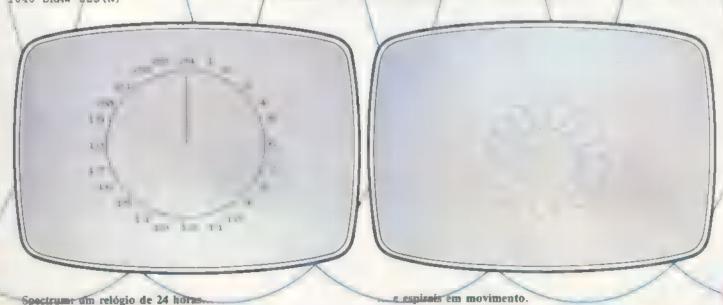
Se compararmos o exemplo acima com os programas, veremos que o laço **FOR...NEXT** que controla o comando **PRINT** é diferente para cada versão. Na verdade, qualquer laço funcionaria em qualquer computador; as diferenças são mantidas por conveniência, conforme o sistema operacional de cada um.

O objetivo da rotina é imprimir números de 1 a 24 ao redor do relógio, marcando as horas. O laço para o programa do Spectrum é:

```
FOR n=1 TO 24
NEXT n
```

A rotina calcula a posição para o **PRINT AT** cada vez que passa pelo laço, dividindo o círculo em 24 segmentos.

O TRS-COLOR e o MSX funcionam como ■ Spectrum mas, como usam uma rotina especial para desenhar letras na

Spectrum: um relógio de 24 horas... ...e espirais em movimento.

tela gráfica, seus programas são um pouco diferentes. Na verdade, eles não imprimem um número relacionado com o STEP que controla suas posições angulares. O que imprimem é acessado — ou chamado — pelo comando **STR$**, na linha que controla a impressão. **STR$** simplesmente consulta a rotina que desenha letras; não está relacionado com a posição onde os números são desenhados, a qual é determinada pela variável **K**.

Como o círculo tem 360 graus, para se imprimir 24 números ao redor do relógio, em intervalos iguais, eles devem ser posicionados a cada 360/24 (15) graus. Do mesmo modo, como há 2*PI radianos num círculo completo, os números estão posicionados a cada 2*PI/24, ou PI/12 radianos.

Observe que em todos os laços o primeiro número é o primeiro salto, e não o 0 (1 no Spectrum, 5 no TRS-Color e MSX). Por isso, o primeiro número impresso — ou desenhado, como é o caso do TRS-Color e do MSX — estará em 1 hora e não em 24 horas.

O restante do programa usa **SIN** e **COS** para calcular as posições dos ponteiros, e uma variável ou o temporizador interno do computador para acompanhar o tempo, garantindo, assim, que o ponteiro de segundos se movimente a cada segundo.

ACESSAR OS DADOS

O método para dizer ao computador o que ele deve imprimir não precisa ser aquele utilizado no exemplo do relógio. Poderíamos armazenar os números em **DATA** e depois ler (**READ**) estes números, um de cada vez, à medida que o computador passasse pelo laço **FOR...NEXT**.

Pode parecer um desperdício de memória mas, na verdade, este método nos permite fazer mudanças interessantes. Por exemplo: existem relógios com algarismos romanos no lugar dos números convencionais. Usando uma variável de cadeia, e armazenando em **DATA** as letras, e não os números, poderemos sofisticar ainda mais nosso relógio.

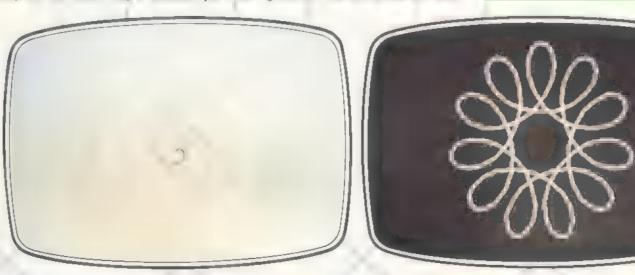
Talvez seja preciso mudar o tamanho e/ou a posição do círculo para que os algarismos romanos caibam na tela (o número 8, por exemplo, é VIII, o que toma bastante espaço!).

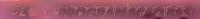
Também poderíamos fazer um relógio de 24 horas em algarismos romanos. E é fácil, usando este método, mudar um relógio de 12 para um de 24 horas.

Adicionar um alarme ao programa também não será difícil. Para isso, precisaríamos calcular, usando SIN e **COS**, a posição exata em que o ponteiro vai estar na hora ajustada para o disparo do alarme. Uma outra linha com um **IF...THEN** avisaria ao computador que a variável que ajusta a posição do ponteiro é igual à calculada para o alarme e, então, o computador emitiria algum som.

Usando os mesmos princípios, seria possível não só fazer um relógio de 12 horas ou de um dia, mas de uma semana, um mês ou até mesmo um ano. O único limite é a quantidade de informação que podemos colocar na tela.

ABSTRATOS

Até agora, empregamos **SIN** e **COS** para coisas úteis como relógios, bússolas e gráficos. Mas talvez um uso mais

Espiral no Spectrum.

Flor no MSX.

interessante esteja na elaboração de gráficos abstratos, obtidos com a manipulação adequada das duas funções.

Desenhando séries de círculos ■ elipses, já produzimos algumas figuras interessantes. Se, por exemplo, movêssemos o centro do círculo ou, então, aumentássemos seu raio ■ cada volta, o resultado seria ainda melhor. Essas mudanças são muito fáceis de fazer: precisaremos apenas acrescentar algumas linhas às rotinas conhecidas.

COMO FAZER UMA ESPIRAL

No artigo anterior, apresentamos uma rotina que desenha um círculo ponto a ponto. Com algumas mudanças — por exemplo, o aumento gradativo do raio —, poderemos obter uma espiral, em vez de um círculo. As versões a seguir fazem exatamente isto.

```
10 LET z=0
50 FOR n=0 TO 8*PI STEP PI/10
60 PLOT 128+(5+z)*SIN n,88+(5
+z)*COS ■
70 LET z=z+1
80 NEXT n
```

```
10 PMODE 4,1
20 PCLS
30 [illegible] 1,1
40 PI=4*ATN(1)
50 FOR N=0 TO 10*PI STEP PI/10
■ PSET(127+(5+Z)*SIN(N),95+(5+
Z)*COS(N),5)
70 Z=Z+1
80 NEXT N
90 GOTO 90
```

```
10  HGR2
20  HCOLOR= 3
30 PI = ■ *  ATN (1)
40  FOR N = 0 TO 8 * PI STEP PI
 / 10
50  HPLOT 140 + (5 + Z) *  SIN
(N),80 + (5 + Z) *  COS (N)
60 Z = Z + 1
70  [illegible] N
80  GOTO 80
```

```
10 SCREEN 2
20 COLOR 1,15
30 CLS
```

```
40 PI=4*ATN(1)
50 FOR N=0 TO 10*PI STEP PI/10
60 PSET(128+(5+Z)*SIN(N),96+(5+
Z)*COS(N)),1
70 Z=Z+1
80 NEXT N
90 GOTO 90
```

A diferença entre este programa e o que desenha um círculo é a variável **Z**. Esta começa em 0, aumentando cada vez que o computador passa pelo laço. A mudança que ela traz está nos cálculos da linha 60 no Sinclair, TRS-Color e MSX e da linha 50 no Apple.

O ponto desenhado é controlado, como no círculo, pelo **SIN** e **COS** da variável de controle no laço. O aumento do número que multiplica o **SIN** e **COS**, quando o computador passa pelo laço, faz com que os pontos sejam desenhados cada vez mais longe do centro. A variável **Z** é responsável pelos consecutivos aumentos, que resultam na espiral crescendo para fora.

Como no programa do círculo, podemos obter resultados diferentes se mudarmos o salto (**STEP**) — linha 40 no Apple e 50 nos outros — ou a dimensão do aumento de **Z**. Alguns **STEP** interessantes são 2, 5,PI e 2*PI. Experimente-os e descubra porque os resultados são diferentes (lembre-se de que os computadores trabalham com radianos, e que um círculo tem 2*PI radianos).

Podemos fazer espirais elípticas do mesmo modo que fizemos elipses na primeira versão deste programa: basta mudar o número dentro dos parênteses na linha 60 (linha 50 no Apple).

SOFISTIQUE OS DESENHOS

Existe um brinquedo infantil que utiliza rodas de plástico para desenhar. A criança coloca uma roda menor dentro de uma maior, e a ponta de uma caneta num dos furos da roda menor; movendo a roda menor pelo lado interno da maior, a caneta gira. O desenho obtido é muito interessante.

O centro da roda menor está sempre se movendo quando a criança está desenhando, o que resulta numa série de espirais contínuas. O computador consegue um efeito semelhante, desenhando círculos cujos centros estão em constante movimento.

Os programas que se seguem fazem isso, sendo que, para o Spectrum, a escolha da cor é aleatória.

```
10 CLS : LET x=0: LET p=0:
LET q=0
20 LET z=INT (RND*7): INK z:
LET a=INT (RND*50)
30 LET b=INT (RND*50)
40 FOR n=0 TO 200*PI STEP b/
100
50 IF INKEY$<>"" THEN  GOTO
10
60 LET p=128+(a-b)*SIN n: LET
q=88+(a-b)*COS n
70 PLOT p+b*SIN x,q+b*COS x
80 LET x=x-a/1074
90 NEXT n
110 GOTO 20
```

```
10 PMODE 4,1:PCLS:SCREEN 1,1
20 PI=4*ATN(1)
30 A=RND(50)-1:B=RND(50)-1
40 FOR N=0 TO 200*PI STEP B/100
50 IF INKEY$<>"" THEN 10
60 P=128+(A-B)*SIN(N):Q=95+(A-B
)*COS(N)
70 PSET (P+B*SIN(X),Q+B*COS(X),
5)
80 X=X-A/1074
90 NEXT
100 GOTO 30
```

```
10  HGR2 : HCOLOR= 3
20 PI = 4 *  ATN (1)
30 A =  INT ( RND (1) * 100)
40 B =  INT ( RND (1) * 40)
50  FOR N = 0 TO 200 * PI STEP
B / 100
60 W =  PEEK ( - 16384)
70  POKE  - 16368,0
80  IF W > 127 THEN 10
90 P = 140 + B *  SIN (N):Q = 8
0 + B *  COS (N)
100  HPLOT P + B *  SIN (X),Q =
 B *  COS (X)
110 X = X - A / 1000
120  NEXT N
130  GOTO 30
```

```
10 SCREEN2:COLOR1,15:CLS
20 PI=4*ATN(1)
30 A=INT(RND(1)*200)+20
40 B=INT(RND(1)*40)+10
50 FOR N=0 TO 200*PI STEP B/100
60 IF INKEY$<>"" THEN 10
70 P=128+B*SIN(N):Q=96+B*COS(N)
80 PSET(P+B*SIN(X),Q+B*COS(X)),
1
90 X=X-A/1000
100 NEXT N
110 GOTO 30
```

Nota-se facilmente que o laço **FOR...NEXT** é uma rotina criadora de círculos: os laços para cada computador são

```
FOR N=0 TO 200*PI
```

Como nos outros programas deste artigo que usam **SIN** e **COS**, essas duas funções também estão inclusas no laço. Elas calculam a posição do próximo ponto a ser desenhado, embora, é claro, os cálculos sejam um pouco mais complicados do que um simples "**SIN N**" ou "**COS N**".

Os demais cálculos nesta linha servem para incrementar o resultado da expressão do **SIN** ou **COS**, para que ele possa representar uma posição na tela. Por essa razão, o centro da tela gráfica de cada computador também faz parte dos cálculos. Se olharmos com atenção, encontraremos o centro da coordenada "X" numa metade da linha e o centro da coordenada "Y" na outra.

O cálculo envolve ainda duas variáveis; como elas recebem valores aleatórios no início de cada programa, este sempre executa desenhos diferentes ao ser rodado. Tente mudar o valor dessas variáveis e observe a diferença.

ALTERAÇÕES

Podemos fazer outro tipo de alteração: mudar o **STEP** do laço **FOR...NEXT**. O resultado será o aumento ou a diminuição da densidade dos pontos, conforme o **STEP** seja aumentado ou diminuído. Para a obtenção do desenho, o centro do círculo deve mudar cada vez que um ponto é desenhado. Para isso, colocamos o próprio centro do círculo num outro círculo, que gira a cada passo do laço. O efeito é produzido pelo segundo grupo de cálculos do **SIN** e **COS**, baseado em x. Essa variável decresce levemente cada vez que o computador passa pelo laço.

Tente alterar o tanto que a variável (x) diminui e veja o que acontece com os desenhos. Estes seis valores trazem resultados interessantes: A/10, A/−10, A/−50, A/0.5, A/0.1, A/0.001.

Todas as versões permitem que reiniciemos o programa a qualquer hora, bastando apertar uma tecla. O Spectrum, o TRS-Color e o MSX usam o **INKEY$** para verificar se alguma tecla foi pressionada, enquanto o Apple busca essa informação na memória (linhas 60 a 80, já comentadas em artigos anteriores sobre **PEEK** e **POKE**).

Examinaremos ainda diferentes usos para funções trigonométricas e angulares. No próximo artigo desta série, trataremos de quadrados, cubos, raízes quadradas e outras ferramentas poderosas para controlar o comportamento das variáveis. Você terá, assim, novas oportunidades de dar às funções empregos que, à primeira vista, não parecem estar relacionados com a matemática.

TRABALHE COM O CÓDIGO ASCII

- O QUE É O CÓDIGO ASCII?
- USE NÚMEROS EM VEZ DE CARACTERES
- UM PROGRAMA PARA CODIFICAR E DECODIFICAR MENSAGENS

Conjunto de códigos alfanuméricos padronizados, o código ASCII foi concebido com o objetivo de tornar possível a transferência de dados de um computador para outro.

Cada tecla ou combinação de teclas do seu micro corresponde a um código eletrônico dentro do computador. Em BASIC, códigos desse tipo são representados por valores decimais entre 0 e 255.

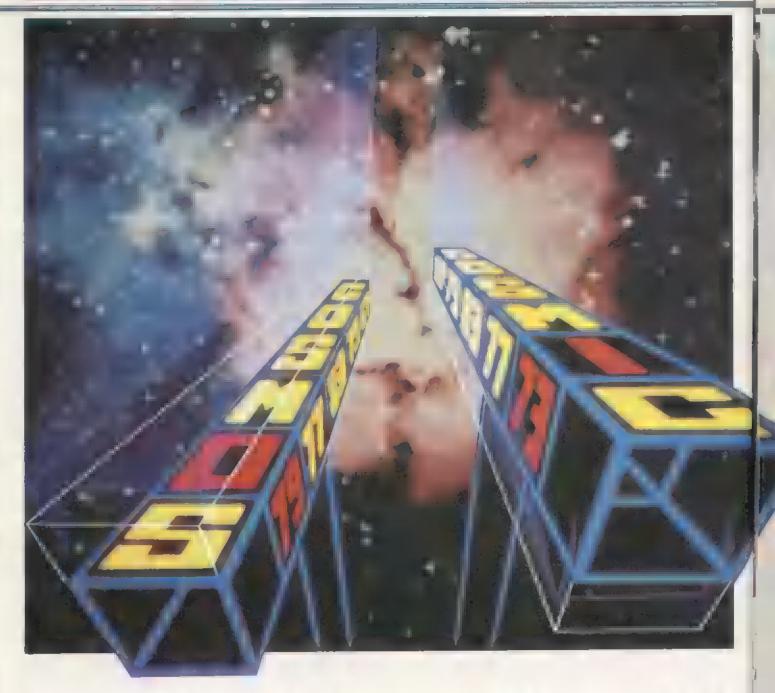
A letra A, por exemplo, tem o código decimal 65, B é 66, e assim por diante. Toda vez que uma letra ou palavra é digitada, o computador [illegible] os códigos correspondentes.

Os valores e caracteres correspondentes não são, porém, os mesmos em todos os computadores; mas felizmente existe alguma padronização no modo com que eles são usados.

Os valores de código são referidos pela sigla ASCII (que vem de *American Standard Code for Information Interchange*, ou seja, Código Padrão Americano para Troca de Informação). Esse código foi montado com o objetivo de possibilitar a transferência de dados de um computador para outro. Os micros TRS, MSX, Apple e Spectrum costumam empregá-lo, pelo menos em parte. Apenas o ZX-81 tem um código completamente diferente.

O [illegible]

O conjunto completo de caracteres ASCII não se apresenta do mesmo modo em todos os computadores. Mas há semelhanças entre os diversos conjun-

tos. A maior delas está na faixa de 33 a 90, que cobre os símbolos mais comuns — pontuação, números e letras maiúsculas. Existe um modo muito simples de descobrir o código ASCII de um caractere: basta digitar **PRINT ASC ("X")** (ou **PRINT CODE "X"** no Spectrum). Você terá, assim, o código de X ou de qualquer outro caractere que for colocado [illegible] seu lugar. O próximo programa facilita as coisas para você:

```
10 PRINT"DIGITE QUALQUER LETRA,
 [illegible] OU CARACTER"
[illegible] INPUT AS
30 PRINT"O CODIGO ASCII [illegible] ";
AS;" E ";ASC(A$)
[illegible] GOTO 20
```

No Spectrum, mude a linha [illegible] para:

```
30 [illegible] "O CODIGO ASCII [illegible]
 ";AS;" [illegible] ";CODE A$
```

O oposto de **ASC** [illegible] **CODE** é **CHR$**, que converte o número de código para o caractere correspondente. O próximo programa converte todos os códigos ASCII de 33 a 90 em caracteres:

```
[illegible] PRINT"CODIGO ASCII","CARACTE
R"
20 [illegible] N=33 TO 90
30 PRINT N,CHR$(N),
40 [illegible] D=1 TO 500:NEXT D
50 NEXT N
```

As letras minúsculas estão na faixa de 97 a 122 e podem ser vistas mudando-se o último número da linha 20 para 122. Ao substituirmos este último por 255,

teremos uma visão do conjunto completo de caracteres. A listagem dos caracteres e códigos correspondentes está no manual do seu micro.

No TRS-Color, os caracteres minúsculos aparecerão na tela iguais aos maiúsculos, só que invertidos. A impressora ligada ■ micro provavelmente interpretará esses caracteres como minúsculos de verdade. O TK-2000 não possui letras minúsculas.

COMO USAR O CÓDIGO ASCII

A vantagem de se usar um número em lugar de uma letra é que se pode alterá-lo matematicamente de várias maneiras. Então, ao se imprimir o **CHR$** do novo número, o que se obtém é uma letra diferente. Base para muitos programas codificadores, essa possibilidade de conversão é muito interessante, visto que não se pode manipular uma letra por meio de regras matemáticas.

Códigos simples apenas adicionam um valor constante ■ cada número, de modo a fazer com que ■ letra "viaje" pelo alfabeto. Por exemplo, a letra A vira G, o B torna-se H, etc. Um código desse tipo é tão fácil de montar quanto de decifrar. Assim, um programa bem simples pode tentar ■ 26 combinações possíveis; um rápido exame determinará qual dessas combinações é a correta.

Para ser útil, contudo, o programa deve fazer coisas mais complicadas. O programa abaixo, por exemplo, usa uma palavra de código que funciona como chave. Assim, cada letra da mensagem é modificada de uma forma diferente. Muito mais difícil de ser decifrado, esse código exige que se conheça a palavra-chave.

```
10 POKE 23658,8: LET CP=0:
LET PM=0: LET D$=""
20 CLS
30 INPUT "QUAL ■ A SENHA? ";
LINE C$
40 PRINT "INTRODUZA A MENSAGE
M "
50 INPUT  LINE M$
60 LET CP=CP+1: IF CP>LEN C$
THEN  LET CP=1
70 LET PM=PM+1
80 IF PM>LEN M$ THEN  GOTO
200
90 LET F$=M$(PM)
100 IF F$<"A" OR F$>"Z" THEN
GOTO 150
110 LET F=CODE F$+CODE C$(CP)-
65
120 IF F>90 THEN  LET F=F-26
130 LET D$=D$+CHR$ F
140 GOTO 60
150 IF F$<"0" OR F$>"9" THEN
LET D$=D$+F$: GOTO 70
160 LET F=CODE F$+CODE C$(CP)-
48
170 IF F>57 THEN  LET F=F-10:
GOTO 170
180 LET D$=D$+CHR$ F
190 GOTO 60
200 PRINT '"A MENSAGEM CODIFIC
ADA E :"
210 PRINT 'D$
220 STOP
```

```
10 CLEAR 300
20 CLS
30 LINEINPUT"QUAL ■ A SENHA?";C
$
40 PRINT"INTRODUZA A MENSAGEM"
50 LINEINPUT ■
60 CP=CP+1:IF CP>LEN(C$)THEN CP
=1
70 PM=PM+1
80 IF PM>LEN(MS$)THEN 200
90 F$=MID$(MS$,PM,1)
100 IF F$<"A" OR F$>"Z" THEN 15
■
110 F=ASC(F$)+ASC(MID$(C$,CP,1)
)-65
120 IF F>90 THEN F=F-26
130 CD$=CD$+CHR$(F)
140 GOTO 60
150 IF F$<"0" OR F$>"9" THEN CD
$=CD$+F$:GOTO 70
160 F=ASC(F$)+ASC(MID$(C$,CP,1)
)-48
170 IF F>57 THEN F=F-10:GOTO 17
0
180 CD$=CD$+CHR$(F)
190 GOTO 60
200 PRINT:PRINT"A MENSAGEM CODI
FICADA E: "
210 PRINT:PRINT CD$
220 ■
```

```
10  HOME
30  INPUT "Palavra chave? ";C$
40  PRINT "Digite ■ mensagem:"
50  INPUT MS$
60 CP = CP + 1: IF CP >  LEN (C
$) THEN CP = 1
70 ■ = PM + 1
80  IF PM >  LEN (MS$) THEN 200
90 F$ =  MID$ (MS$,PM,1)
100  IF F$ < "A" OR F$ > "Z" TH
■ 150
110 F =  ASC (F$) +  ASC ( MID$
(C$,CP,1)) - 65
120  IF F > 90 THEN F = F - 26
130 CD$ = CD$ +  CHR$ (F)
140  GOTO 60
150  IF F$ < "0" OR F$ > "9" TH
EN CD$ = CD$ + F$: GOTO 70
160 F =  ASC (F$) +  ASC ( MID$
(C$,CP,1)) - 48
170  IF F > 57 THEN ■ = F - 10:
GOTO 170
180 CD$ = CD$ +  CHR$ (F)
190  GOTO 60
200  PRINT : PRINT "A mensagem
codificada e: "
210  PRINT : PRINT CD$
220  END
```

```
20 CLS
30 LINEINPUT"Palavra de código?
";C$
40 PRINT"Digite a mensagem:"
50 LINEINPUT MS$
60 CP=CP+1:IFCP>LEN(C$)THENCP=1
70 PM=PM+1
■ IFPM>LEN(MS$)THEN200
90 F$=MID$(MS$,PM,1)
100 IFF$<"A"ORF$>"Z"THEN150
110 F=ASC(F$)+ASC(MID$(C$,CP,1)
)-65
120 IFF>90THENF=F-26
130 CD$=CD$+CHR$(F)
140 GOTO60
150 IFF$<"0"ORF$>"9"THENCD$=CD$
+F$:GOTO70
160 F=ASC(F$)+ASC(MID$(C$,CP,1)
)-48
170 IFF>57THENF=F-10:GOTO170
180 CD$=CD$+CHR$(F)
190 GOTO60
200 PRINT:PRINT"A mensagem codi
ficada é:"
210 PRINT:PRINTCD$
220 END
```

Depois de pedir que sejam digitadas a palavra-chave e a mensagem, ■ programa faz a codificação, mostrando-a na tela. A mensagem ficará então preservada (ou seja, somente alguém que conheça o código poderá decifrá-la).

O modo como isso funciona é bastante simples, assemelhando-se muito ao método descrito anteriormente. A diferença está no fato de que, em vez de adicionar um valor fixo ■ cada letra, o código ASCII da primeira letra da palavra-chave é acrescentado à primeira letra da mensagem; ■ segunda letra da palavra-chave é adicionada ■ segunda da mensagem, ■ assim por diante. Quando o fim da palavra-código é encontrado, o ciclo recomeça.

Se ■ resultado de algumas das adições for maior do que 90 (o que significa que a letra correspondente ficará situada além do Z), devemos subtrair 26, de modo a trazer o caractere para dentro do alfabeto. Os números são manipulados separadamente nas linhas 150 ■ 180 para manter o código entre 48 e 57, que corresponde ■ 0 e 9.

O programa decodificador é muito semelhante ao primeiro, diferindo apenas em alguns sinais + e – que foram mudados, além de certos valores. Os números das linhas não coincidem; assim,

pode-se usar o programa combinado com o anterior. Algumas linhas foram colocadas de modo que se possa escolher entre codificar e decodificar uma mensagem.

Se você está familiarizado com os comandos de remuneração e edição de linhas do seu computador, tente usá-los para transformar o programa anterior no programa que apresentamos agora. Depois, junte-os (MERGE).

```
 12 INPUT "(C)ODIFICAR [illegible] (D)E
 CODIFICAR?"; LINE A$
 14 IF A$="D" THEN  GOTO 400
 16 IF A$<>"C" [illegible]  GOTO 12
400 CLS
410 INPUT "QUAL E A [illegible] ";
LINE C$
420 PRINT "INTRODUZA [illegible] MENSAGE
M CODIFICADA "
430 INPUT  LINE M$
440 LET CP=CP+1: IF CP>LEN C$
THEN  LET CP=1
450 LET PM=PM+1
[illegible] IF PM>LEN M$ THEN  GOTO
580
470 LET F$=M$(PM)
[illegible] IF F$<"A" [illegible] F$>"Z" THEN
GOTO 530
490 LET F=CODE F$-CODE C$(CP)+
[illegible]
500 IF F<65 THEN  LET F=F+26
510 LET D$=D$+CHR$ F
520 GOTO 440
530 IF F$<"0" OR F$>"9" THEN
LET D$=D$+F$: GOTO 450
540 LET F=CODE F$-CODE C$(CP)+
[illegible]
550 IF F<48 [illegible]  LET F=F+10:
GOTO 550
560 LET D$=D$+CHR$ [illegible]
570 GOTO 440
580 PRINT '"A MENSAGEM DECODIF
ICADA E "
590 PRINT 'D$
600 STOP
```

```
12 INPUT (C)ODIFICAR OU (D)ECOD
IFICARPRINT:A$
14 IF A$="D" THEN GOTO 400
16 IF A$<>"C" [illegible] GOTO 12
[illegible] CLS
410 LINEINPUT"QUAL E [illegible] SENHA?";
C$
420 PRINT"INTRODUZA [illegible] [illegible]
CODIFICADA"
430 LINEINPUT MS$
440 CP=CP+1:IF CP>LEN(C$) [illegible]
CP=1
450 PM=PM+1
[illegible] IF PM>LEN(MS$)THEN 580
470 F$=MID$(MS$,PM,1)
[illegible] IF F$<"A" OR F$>"Z" [illegible] 53
[illegible]
490 F=ASC(F$)-ASC(MID$(C$,CP,1)
)+65
500 IF F<65 [illegible] F=F+26
```

```
510 CD$=CD$+CHR$(F)
520 GOTO 440
530 IF F$<"0" [illegible] F$>"9" THEN CD
$=CD$+F$:GOTO 450
540 F=ASC(F$)-ASC(MID$(C$,CP,1)
)+48
550 IF F<48 THEN F=F+10:GOTO 55
0
560 CD$=CD$+CHR$(F)
570 GOTO 440
580 PRINT"A [illegible] DECODIFICA
DA E: "
590 PRINT:PRINT CD$
600 END
```

```
12  INPUT "[C]odificar [illegible] [D]ec
odificar ";A$
14  IF A$ = "D" THEN 410
16  IF A$ <  > "C" THEN 12
400  [illegible]
410  INPUT "Palavra chave? ";C$

420  PRINT "Digite a mensagem:"

430  INPUT MS$
440 CP = CP + 1: IF CP >  LEN (
C$) THEN CP = 1
450 PM = PM + 1
460  IF PM >  [illegible] (MS$) THEN [illegible]
0
470 F$ =  MID$ (MS$,PM,1)
480  IF F$ < "A" [illegible] F$ > "Z" TH
[illegible] 530
490 F =  ASC (F$) -  ASC ( MID$
 (C$,CP,1)) + 65
500  IF F < 65 THEN F = F + 26
510 CD$ = CD$ +  CHR$ (F)
520  GOTO 440
530  IF F$ < "0" [illegible] F$ > "9" TH
[illegible] CD$ = CD$ + F$: GOTO 450
540 F =  ASC (F$) -  ASC ( MID$
 (C$,CP,1)) + 48
550  IF F < [illegible] THEN F = F + 10:
 GOTO 550
560 CD$ = CD$ +  CHR$ (F)
570  GOTO 440
[illegible]  PRINT : PRINT "A mensagem
decodificada e: "
590  PRINT : PRINT CD$
600  END
```

```
12 INPUT"(C)odificar [illegible] (D)ecod
ificar? ";A$
14 IF A$="D"THEN400
16 IF A$<>"C"THEN12
400 CLS
410 LINEINPUT"Palavra de código
? ";C$
420 PRINT"Digite a mensagem:"
430 LINEINPUT MS$
440 CP=CP+1:IFCP>LEN(C$)THENCP=
1
450 PM=PM+1
[illegible] IFPM>LEN(MS$)THEN580
470 F$=MID$(MS$,PM,1)
480 IFF$<"A"ORF$>"Z"THEN530
[illegible] F=ASC(F$)-ASC(MID$(C$,CP,1)
```

```
)+65
500 IFF<65THENF=F+26
510 CD$=CD$+CHR$(F)
520 GOTO440
530 IFF$<"0"ORF$>"9"THENCD$=CD$
+F$:GOTO450
540 F=ASC(F$)-ASC(MID$(C$,CP,1)
)+48
550 IFF<48THENF=F+10:GOTO550
560 CD$=CD$+CHR$(F)
570 GOTO440
580 PRINT:PRINT"A mensagem deco
dificada é:"
590 PRINT:PRINTCD$
600 END
```

Se você quer que o seu código seja realmente indecifrável, codifique a mensagem usando uma segunda palavra-chave. Desse modo, ele se tornará à prova de espiões. Mas tenha o cuidado de não trocar a ordem das duas palavras-chave na hora de decodificar. Do contrário, você poderá causar uma confusão monumental.

COMPARAÇÕES

Uma das funções mais importantes do código ASCII consiste em facilitar a comparação de palavras (veja página 241 e seguintes).

Essa operação é feita letra por letra pelo computador.

Suponhamos, por exemplo, que se queira comparar as palavras COSMONAUTA e COSMOPOLITA. O computador começará confrontando os dois C; em seguida, ele passará sucessivamente para os dois O, os dois S, os dois M, os dois O, até chegar ao N e ao P.

Vale lembrar que o que está sendo comparado não são posições alfabéticas arbitrárias, mas os códigos ASCII de cada caractere.

Como P tem um valor ASCII maior que N, nesta comparação, a palavra COSMOPOLITA tem um valor maior do que a expressão COSMONAUTA. Note que nem a soma dos códigos nem o número de letras são levados em conta.

Cada letra é comparada individualmente com a sua correspondente (para um aprofundamento do tema, veja outros exemplos de comparações de palavras e cordões no artigo *Cadeia de Caracteres*, página 241).

A seguir, apresentamos alguns exemplos de comparações de vários valores possíveis para A$ e B$, juntamente com os códigos ASCII envolvidos. É importante comparar os valores do código de cada caractere aos pares: o primeiro caractere de A$ com o primeiro de B$ e assim por diante, até o fim da palavra mais curta.

A$	ASCII	B$	ASCII	RELAÇÃO
ABC	65,66,67	ABC	65,66,67	A$ = B$
ABD	65,66,68	ABCD	65,66,67	A$ > B$
ABD	65,66,68	ABCD	65,66,67	A$ < > B$
ABC	65,66,67	Abc	65,97,98	A$ < B$
COSMI	67,79,83	COSMO	67,79,83	A$ < B$
	77,73		77,79	
$1	36,49	$1.0	36,49,46	A$ < B$

Note que no segundo e terceiro exemplos é possível escrever a relação de várias maneiras. No último exemplo, como todos os caracteres são iguais, a palavra mais longa é a de maior valor.

VERIFIQUE AS ENTRADAS

As funções **ASC** e **CODE** funcionam apenas no primeiro caractere de uma variável alfanumérica. Assim, **PRINT ASC ("BRASIL")** — ou **PRINT CODE "BRASIL"**, no Spectrum — mostrará 65, o código de B. Isto é muito útil para ■ checar comandos **INPUT** do programa. Veja, por exemplo, o programa anterior. A linha 12 pede que você digite um "C" para codificar, ou um "D" para decodificar. As duas linhas seguintes direcionam o programa da forma adequada. Mas, para evitar que o computador não aceite as entradas **CODIFICAR** ou **DECODIFICAR**, escritas por extenso, pode-se reescrever as linhas 14 e 16 como se segue:

```
14 IF ASC(A$)-68 THEN GOTO 400
16 IF ASC(A$)<>67 THEN GOTO 12
```

```
14 IF CODE A$=68 THEN  GOTO
400
16 IF CODE A$<>67 THEN  GOTO
12
```

CÓDIGOS DE CONTROLE

Alguns códigos ASCII não têm caracteres associados. Quando você tecla <ENTER> ou <RETURN> o micro identifica o valor 13; mas, em vez de imprimir um caractere na tela, o cursor é colocado no início da linha seguinte; ou, como acontece no Spectrum, o programa é listado. O código 13 é chamado de código de retorno do carro (*Carriage Return*; daí o <CR> dos modelos Apple). Experimente colocar no computador **PRINT "A"; CHR$ (13); "B"**. O "B" será impresso ■ linha abaixo do "A".

Este programa permite ver os códigos de controle:

```
10 PRINT CODE INKEY$
20 GOTO 10
```

```
5 A$=INKEY$:IF A$="" THEN 5
10 PRINT ASC(A$)
20 GOTO 10
```

```
5  GET A$
10  PRINT  ASC (A$)
20  GOTO 5
```

Tente teclar <ENTER> ou <ESC> ou pressionar a tecla de retrocesso, ou qualquer outra do teclado, sozinha ou simultaneamente com <SHIFT> ou <CTRL>, e vá descobrindo os códigos que seu micro utiliza.

Alguns computadores usam mais esses códigos do que outros. Afinal, existem muitos números disponíveis entre 0 e 31 e alguns acima de 90. Tais códigos foram originalmente definidos quando os computadores trabalhavam apenas com ■ antigas impressoras. Atualmente, o computador é ligado a um televisor ou monitor de vídeo e muitos desses códigos tornaram-se obsoletos.

O Spectrum teve muitos de seus antigos códigos redefinidos, de forma ■ trabalhar com a tela de TV. A instrução **PAPER** tem código 17 e **INK**, 16. Assim, para escrever a palavra "TÍTULO" em vermelho sobre fundo verde, é preciso digitar o seguinte:

```
10 LET A$=CHR$ 17+CHR$ 4+CHR$
16+CHR$ 2+"TITULO"
20 PRINT A$
```

Usado muitas vezes em um programa, esse cabeçalho precisa ser definido apenas uma vez. Sempre que se quiser usá-lo basta digitar **PRINT A$**.

O Apple também usa caracteres de controle, tanto a nível operacional como ■ nível de programas. Assim, você deve conhecer muito bem ■ **CTRL-S**, o **CTRL-C** ou o **CTRL-D**. Nesse computador, os códigos dos caracteres de controle de **CTRL-A** até **CTRL-Z** ocupam os números de 1 a 26. Desta forma, **CHR$ (1)** equivale a **CTRL-A**; **CHR$ (2)**, a **CTRL-B**, etc.

CÓDIGOS PARA O MSX

- O QUE SÃO OS CÓDIGOS DE CONTROLE
- COMO UTILIZÁ-LOS EM PROGRAMAS BASIC
- MENSAGENS COM SONS

Muito úteis para uma série de truques de programação, os códigos de controle ocupam o espaço que vai de 0 a 31 no código ASCII. Aprenda a usá-los nos micros MSX.

Os códigos de controle correspondem, na maioria dos microcomputadores, aos números entre 0 e 31 no sistema ASCII, e servem para realizar uma série de funções relacionadas principalmente com o controle do vídeo. Em artigos anteriores desta série, mostramos como os diferentes fabricantes de micros aproveitam a faixa de códigos de 0 a 31 para implementar essas funções de controle. Ao contrário do espaço de código ASCII que vai de 32 em diante, não há nessa faixa qualquer padronização entre os diversos modelos de computadores. Vejamos agora como os usuários de micros da linha MSX podem explorar os seus códigos de controle.

A FUNÇÃO CHR$

Embora o MSX disponha de uma tecla <CONTROL>, que serve para gerar os códigos de 0 a 31 (por exemplo, pressionando-se as teclas <CONTROL> e <A> simultaneamente obtém-se o código 1), um programa BASIC só pode acionar os códigos de controle por intermédio da função **CHR$** (abreviatura de *character*). Essa função permite converter um código numérico inteiro entre 0 e 255 em seu caractere correspondente. Por exemplo, **PRINT CHR$ (65)** escreverá na tela a letra A.

Usando o **PRINT CHR$** dessa forma, podemos controlar diversas funções do vídeo, por meio dos códigos de 0 a 31. A seguir, apresentamos os códigos que proporcionam os efeitos mais interessantes no MSX:

1 - determina caractere gráfico
7 - aciona o alarma sonoro
8 - apaga o caractere antes do cursor
9 - tabula
10 - pula uma linha (**LINEFEED**)
11 - coloca o cursor na posição 1,1 (**HOME**)
12 - equivale ao **CLS**
13 - retorno de carro (**RETURN**).

Ao contrário dos códigos ASCII de 32 em diante, a função **CHR$** associada aos códigos de 0 a 31 não imprime qualquer caractere visível na tela, causando apenas um efeito imediato.

O programa a seguir mostra uma aplicação interessante dos códigos de controle. Ele está estruturado de tal modo que, toda vez que uma mensagem de erro for mostrada na tela, o computador acionará o alarma sonoro interno (bipe) e pulará uma linha em branco antes da mensagem. Em vez de repetir a mesma seqüência de comandos para cada mensagem de erro, definimos uma variável **C$**, que contém os códigos 7 e 10. Depois, basta usar o **PRINT** da mensagem, precedendo-a com essa variável.

```
10 C$=CHR$(10)+CHR$(7)+"*** ERR
O: "
20 CLS:PRINT "Cálculo da raiz q
uadrada"
30 PRINT
35 X$=""
40 LINE INPUT "ENTRE UM NUMERO:
";X$
45 IF X$="" THEN END
50 X=VAL(X$)
60 IF X=0 THEN PRINT C$;"Entrad
a deve ser um número":GOTO 40
70 IF X<0 THEN PRINT C$;"Numero
 deve ser maior do que zero":GO
TO 40
80 PRINT
90 PRINT "A raiz quadrada de";X
;"é igual a";SQR(X)
100 PRINT
110 GOTO 40
```

COMO PASSEAR PELO VÍDEO

Outra aplicação interessante dos códigos de controle é a manipulação do cursor de texto. Assim, os códigos 28 a 31 movimentam o cursor da seguinte maneira:

28 - move o cursor para a direita
29 - move o cursor para a esquerda
30 - move o cursor para cima
31 - move o cursor para baixo.

Um pequeno programa, que ilustra bem o uso destes códigos é mostrado a seguir:

```
5 CLS
10 LOCATE 10,15
20 FOR I=1 TO 1000
30 R=INT(RND(1)*4)
35 PRINT CHR$(28+R);".";CHR$(29
);
40 NEXT I
```

Este é um curtíssimo exemplo do chamado "passeio aleatório em duas dimensões". Com a execução do programa, o cursor passará a se deslocar pela tela nas quatro direções possíveis.

As linhas 5 e 10 limpam a tela e colocam o cursor em seu centro. O laço que vai das linhas 20 a 40 coloca 1 000 pontos aleatoriamente na tela. A linha 30 sorteia um número entre 0 e 3. A linha 35 movimenta o cursor com a ajuda da função **CHR$(28 + R)** (que gera um número entre 28 e 31), imprime um ponto e recua o cursor de novo com a função **CHR$(29)**.

A TECLA DE CONTROLE

Como foi explicado, a tecla <CONTROL> serve apenas para acionar os códigos de controle diretamente a partir do teclado (ou seja, não é possível "enxertar" códigos de controle dentro de mensagens entre aspas, como é feito com os micros da linha Apple). Por outro lado, diversas teclas já existentes no MSX para limpeza de tela, controle do cursor, etc. são duplicadas por uma combinação de <CONTROL> com outra tecla. Por exemplo, <CONTROL> <I> avança uma posição de tabulação na tela, e equivale à tecla <TAB>.

Entretanto, algumas funções — como o apagamento total ou parcial de uma linha, o avanço do cursor até a palavra seguinte, etc. — não têm teclas correspondentes, e podem ser usadas com vantagem na edição de programas.

Para gerar um código de controle por meio da tecla <CONTROL>, basta lembrar-se da seguinte relação: o código 1 corresponde à letra A, o código 2 corresponde à letra B, e assim por diante. Por exemplo, para fazer o cursor descer uma linha (caractere de controle chamado **LINEFEED**, **LF**, ou "alimentação de linha"), basta pressionar conjuntamente as teclas <CONTROL> e J.

UM JOGO DE TIRO AO PATO

Pratique tiro ao alvo, caçando patos ... no computador. Mas, ■ vida real, respeite o direito dos bichinhos à existência, deixando-os saborear os doces frutos da liberdade.

Se você leu o artigo anterior de *Programação de Jogos*, já deve ter gravado em fita o programa que movimenta uma alça de mira pela tela com auxílio do joystick. No entanto, embora seja interessante ver como se faz isso em BASIC, o programa não terá muito sentido se não houver um alvo.

Agora, mostraremos como incluir a sub-rotina de controle do joystick em programas de jogos. Isso ■ feito acrescentand■-se ao programa original os subprogramas apresentados ■ seguir. Dessa forma, obteremos um jogo de tiro ■ pato.

Um de cada vez, dez patos aparecerão por um breve momento na tela. O objetivo do jogo é alvejá-los antes que eles desapareçam. A contagem de pontos depende da velocidade e da pontaria do jogador; a cada tiro perdido são descontados alguns pontos. A esse desconto daremos ■ nome de multa.

Carregue ou digite o programa anterior no computador ■ acrescente as linhas a seguir (evidentemente, estas variarão de acordo com o tipo de computador ao qual estão destinadas).

S

Acrescente as próximas linhas ao programa que movimenta o joystick.

```
120 LET hs=0
125 LET i=0: PRINT  PAPER 2;
INK 7;" SCORE";TAB 14;"RECORDE
";TAB 31;" "
135 PRINT  PAPER 3; OVER 0;AT
0,23;hs
145 LET s=0
150 FOR n=1 TO 10
160 LET dx=INT (RND*31)
170 LET dy=INT (RND*20)+1
180 PRINT  ■■■ 6;AT dy,dx;CHR$
144;CHR$ 145;AT dy+1,dx;CHR$
146;CHR$ 147
190 POKE 23672,0: POKE 23673,0
205 PRINT  OVER 0; PAPER 3;AT
0,8;s;" "
210 IF i<>48 THEN  GOTO 400
220 IF ATTR (y,x)=14 AND ATTR
(y+1,x+1)=14 THEN  LET s=s+260
-(PEEK 23672): SOUND .02,30:
GOTO 410
230 LET s=s-20: LET i=0
400 IF PEEK 23572+256*PEEK
23673<240 THEN  GOTO 200
410 PRINT  ■■■ 7;AT dy,dx; CHR$
144;CHR$ 145;AT dy+1,dx;CHR$
146;CHR$ 147
420 NEXT n
430 IF s>hs THEN  LET hs=s
440 PRINT  OVER 0; PAPER 3;AT
0,8;s;AT 0,23;hs
450 PRINT  OVER 0; PAPER 3;
FLASH 1;AT 10,2;"QUALQUER TECL
A PARA RECOMECAR"
460 FOR n=1 TO 100: NEXT ■
470 IF INKEY$="" THEN  GOTO
470
480 CLS : GOTO 125
```

As linhas 120, 125 e 135 cuidam da contagem de pontos. Como o jogo apenas começou, ■ recorde do momento é zero; ele é mostrado junto com o score. A variável i, de entrada do joystick, também torna-se zero.

O laço **FOR...NEXT** entre as linhas 150 e 420 coloca na tela uma série de dez patos que o jogador tentará acertar. A cada volta do laço, as linhas 160 e 170 escolhem uma nova posição para o pato. A linha 180 coloca a ave na tela, imprimindo os quatro caracteres (UDG) que formam o desenho.

A linha 190 coloca o valor zero no contador de tempo. Este será usado para medir os segundos que o pato permanecerá na tela. Quando esse tempo se esgotar, o pato será desenhado em outra posição.

Depois que o programa retornar da sub-rotina de controle do joystick, a linha 205 apagará o primeiro caractere do score. Esta é uma precaução para quando o score estiver diminuindo; assim, nenhuma parte do score anterior permanecerá na tela.

O joystick do Atari tem um botão para atirar; quando acionado, esse botão envia ao computador o caractere de código 48. A linha 210 detecta o tiro, mas só com a mira parada. É possível verificar se foi dado um tiro com a mira em movimento, mas isto ocuparia mais quatro comandos **IF** — um para cada direção. Não é aconselhável fazer isso em BASIC porque diminuiríamos a velocidade do programa.

Caso um tiro tenha sido disparado, o programa verificará se o pato foi atingido, usando **ATTR** na linha 220. **ATTR** verifica a cor de determinada porção da tela. Se a cor for a do pato, o tiro será considerado certeiro, e o jogador ganhará pontos. O cálculo desses pontos é baseado no tempo gasto pelo jogador para acertar o pato. Cada tiro bem-sucedido é anunciado por um bip do computador.

Se **ATTR** não fornecer ■ cor do pato, o programa continuará na linha 230, onde uma multa de vinte pontos será subtraída do total, e o valor de **i**, igualado a zero para que ■ multa não seja cobrada novamente (o que só acontecerá quando outro tiro for desperdiçado). A linha 400 verifica se o limite de tempo foi ultrapassado, saltando para a linha 200, ■■ isto ainda não tiver acontecido. Caso mais de cinco segundos tenham se passado, a linha 410 fará o pato desaparecer.

Depois que os dez patos forem colocados na tela, o programa alcançará a linha 430, que fará ■ comparação entre o score do jogador e o recorde atual. Se for o caso, o recorde será atualizado. O recorde e o score são mostrados pela linha 440. O programa termina com a frase usual: "Qualquer tecla para recomeçar".

O nível de dificuldade do jogo pode ser alterado modificando-se, na linha 400, o limite de tempo que o jogador tem para acertar o pato. Valores compatíveis para ■ conquista de pontos e para a multa devem ser colocados nas linhas 220 e 230.

Não há programa de tiro ao pato para o ZX-81, mas as modificações que ■■ seguem permitirão que você movimente o monstro do programa da página 316 usando o joystick. O botão do joystick provoca ■ inversão da tela (você deve usar a versão completa do programa).

```
115 LET A=CODE A$
120 IF A=33 ■■■ X>0 THEN LET X=
X-1
130 IF A=36 AND X<28 THEN LET ■
=X+1
140 IF A=35 AND Y>0 THEN LET Y=
Y-1
150 IF A=34 AND Y<20 THEN LET Y
=Y+1
155 IF A=28 THEN RAND USR 16606
```

28 é o código enviado ao computador pelo botão do joystick.

MSX

O programa do MSX é completado pelas seguintes linhas e modificações.

```
10 SCREEN 1,2:KEY OFF:COLOR 15,
1,10:R=RND(-TIME)
20 FOR I=1 TO 64
55 SPRITE$(1)=RIGHT$(A$,32,
70 FOR I=4 TO 10
80 VPOKE BASE(6)+I,15*16+1
90 NEXT
100 VPOKE BASE(6)+30,1*16+1
105 VPOKE BASE(6)+31,6*16+6
110 CLS:STRIG(1) ON:SC=0
120 GOSUB 4000
130 FOR I=0 TO 32:VPOKE BASE(5)
+I,255:NEXT
140 FOR I=1 TO 22:VPOKE BASE(5)
+31+I*32,255:VPOKE BASE(5)+32+I
*32,255:NEXT
150 FOR I=0 TO 31:VPOKE BASE(5)
+23*32+I,255:NEXT
160 GOSUB 3000
180 D=10
190 TIME=0
210 PUT SPRITE 1,(DX,DY),10
220 ON STRIG GOSUB 2000,2000,20
00,2000,2000:IF D<1 THEN 250
230 IF TIME<500 THEN 200
240 D=D-1:IF D>0 THEN GOSUB 300
0:GOTO 190
250 GOSUB 4000
260 FOR I=257 TO 286:VPOKE BASE
(5)+I,245:NEXT
280 LOCATE 1,2:PRINT "QUER JOGA
R NOVAMENTE (S/N) ?"
290 A$=INKEY$:IF A$<>"S" AND A$
<>"N" THEN 290
300 IF A$="S" THEN 110
310 SCREEN 0:COLOR 15,4,4:END
2000 IF ABS(X-DX)>1 AND ABS(Y-D
Y)>1 THEN 2070
2010 PUT SPRITE 1,(DX,209)
2020 PLAY "T255O6CL32EG"
2030 GOSUB 3000
2040 SC=SC+550-TIME:GOSUB 4000
2050 D=D-1:TIME=0
2060 RETURN
2070 PLAY "T255O1DDE"
2080 SC=SC-25:GOSUB 4000
2090 RETURN
3000 DX=INT(RND(1)*220+10):DY=I
NT(RND(1)*145+25):RETURN
4000 FOR I=33 TO 62:VPOKE BASE(
5)+I,245:NEXT
4010 LOCATE 0,1:PRINT "SCORE:";
:LOCATE 7,1:PRINTSC;
4020 IF SC>HI THEN HI=SC
4030 LOCATE 14,1:PRINT "RECORDE
:";:LOCATE 24,1:PRINT HI;
4040 RETURN
5000 DATA  3 , 12 , 16 , 32 , 3
2 , 64 , 64 , 127 , 64 , 64 , 3
2 , 32 , 16 , 12 , 3 , 0 , 224
, 152 , 132 , 130 , 130 , 129 ,
 129 , 255 , 129 , 129 , 130 ,
130 , 132 , 152 , 224 , 0
5010 DATA  14, 27, 127, 31, 15,
 7, 15, 31, 31, 29, 30, 15, 3,
1, 1, 3, 0, 0, 0, 0, 0, 192, 11
2, 188, 206, 30, 124, 248, 224,
 64, 64, 224
```

A linha 10 foi modificada para mudar as cores da tela e iniciar o gerador de números aleatórios.

A linha 20 foi alterada para poder ler os dados adicionais da linha 5010. O sprite do pato é criado pela linha 55.

As linhas 70 a 105 mudam a tabela de cores — **BASE (6)** — com o objetivo de alterar as cores dos caracteres. Isto não é essencial ao nosso programa, mas permitirá que você modifique os valores **P** e **T** do comando **VPOKE BASE (6) + I, T * 16,P** (**P** é a cor de fundo e **T**, a cor de frente).

A linha 110 apaga a tela e liga a função **STRIG (1)** que permite a detecção de um tiro (ela também reduz o score a zero). Quando o botão de tiro do joystick 1 é pressionado, a função **STRIG (N) ON** desvia o curso do programa para o endereço do comando **ON STRIG GOSUB**. **N** determina o joystick: 1 e 3, joystick da direita; 2 e 4, joystick da esquerda. O número 0 permite à tecla de espaços disparar tiros.

A linha 120 chama a sub-rotina 4000, que coloca o score e o recorde na tela.

As linhas 130 a 150 desenham uma moldura vermelha, colocando na tela uma série de caracteres com **VPOKE**. A cor da moldura é determinada pela linha 105.

A posição em que o pato vai aparecer é escolhida ao acaso pela linha 160. As linhas 180 e 190 estabelecem as condições iniciais: dez patos na variável **D**, score zero e contador de tempo — **TIME**, uma variável interna do MSX — também igual a zero.

A linha 210 desenha o pato. A linha 220 diz ao programa para onde ir, caso o botão de tiro seja pressionado. São necessários cinco endereços após **ON STRIG** para cobrir os valores de 0 a 4. Se, ao retornar da sub-rotina 2000, onde é verificado se o tiro atingiu o alvo, não houver mais patos — ou seja, **D < 1** — o programa irá para a linha 250.

A linha 230 verifica se o tempo que cada pato pode ficar na tela foi ultrapassado. Se isto ainda não aconteceu, o programa voltará à linha 200.

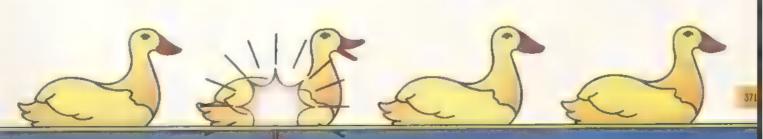
Caso o pato não seja atingido dentro do prazo, a linha 240 diminuirá o número de aves. Se ainda houver patos, a nova posição será calculada pela sub-rotina 3000 e o programa voltará para a linha 190.

Quando não houver mais patos, a linha 250 chamará a sub-rotina 4000, que mostrará o placar.

A linha 280 faz então a velha pergunta: "Quer jogar novamente?" (a linha 260 apaga a região onde é escrita a mensagem). Caso a resposta seja afirmativa, a linha 300 recomeçará o jogo. Caso seja negativa, a linha 310 terminará com ele. A linha 290 cuida das outras respostas.

A sub-rotina 2000 verifica se o tiro dado acertou o pato. Na linha 2000, as condições **ABS (X-DX) > 1** e **ABS (Y-DY) > 1** dão uma certa margem de erro à posição da mira. Se exigirmos que as coordenadas do pato — **DX, DY** — sejam exatamente iguais às coordenadas da mira — **X, Y** —, o jogo ficará muito difícil.

A linha 2010 faz com que o sprite do pato desapareça da tela, pois torna sua coordenada vertical igual a 209. Uma pequena melodia celebra o sucesso do jogador, na linha 2020. A linha 2030 chama a sub-rotina 3000 que calcula uma nova posição para o pato.

Os pontos a que o jogador fez jus são calculados pela linha 2040 com base no tempo que ele levou para acertar o pato. A sub-rotina 4000 mostra o novo placar. A linha 2050 reduz em 1 o número de patos e faz o contador de tempo começar novamente do 0.

Caso o jogador erre o tiro, a linha 2070 produzirá algumas notas dissonantes. Uma multa de 25 pontos será então cobrada e o placar, modificado e mostrado ao jogador.

A linha 3000 chamada como sub-rotina calcula ao acaso a nova posição do pato.

A sub-rotina que começa na linha 4000 mostra na tela o score, atualizando e exibindo o recorde.

Os valores da linha 5010 contêm o padrão do sprite do pato.

O nível de dificuldade do jogo pode ser modificado alterando-se o prazo que o jogador tem para acertar o pato. Isso é feito na linha 230. Acompanhando essa alteração, o cálculo dos pontos (na linha 2040) e da multa (na linha 2080) deve sofrer modificações proporcionais.

Complete o programa anterior com as linhas:

```
60  HOME : [illegible] : SCALE= 1: ROT=
 0:X = 130:Y = 90
100 DX =  INT ( RND (1) * 256):
DY =  INT ( RND (1) * 138)
110 D = 10:SC = 0
120 CV = 0
210  HCOLOR= 3: GOSUB 700
220  IF  PEEK ( - 16287) > 127
OR  PEEK (16286) > 127 THEN  GO
SUB 1000: IF D < 1 THEN 250
230  IF CV < 40 THEN 200
240 D = D - 1: IF [illegible] > 0 THEN  G
OSUB 1500: GOTO 120
250 : HOME : HTAB 5: VTAB 23: P
RINT "SCORE = ";SC;
260  IF SC > HI THEN HI = SC
270  PRINT  TAB( 20);"RECORDE =
 ";HI
280  PRINT  TAB( 5);"QUER JOGAR
 NOVAMENTE (S/N) ? "
290  GET A$: IF A$ <  > "S" AND
 A$ <  > "N" THEN 290
300  IF A$ = "S" THEN 60
310  HOME : TEXT : END
500 LX = X:LY = Y:CV = CV + 1
700  DRAW 1 AT DX,DY
710  DRAW [illegible] AT DX,DY + 8
720  [illegible] 3 AT DX + 8,DY
730  DRAW 4 AT DX + 8,DY + [illegible]
740  RETURN
1000  IF  ABS (X - DX - 4) > 8
OR  ABS (Y - DY - 4) > 8 THEN 1
070
1010  FOR I = 1 TO 20
1020 [illegible] =  PEEK ( - 16336): NEXT

1030  GOSUB 1500
1040 SC = SC + 50 - CV
1050 D = D - 1:CV = 0
1060  RETURN
1070  FOR I = 1 TO 3
1080 P =  PEEK ( - 16336): NEXT

1090 SC = SC - 5
1100  RETURN
1500  HCOLOR= 0: GOSUB 700
1510  HCOLOR= 3: DRAW 5 AT X,Y
1520 DX =  INT ( RND (1) * 256)
:DY =  INT ( RND (1) * 138)
1530  RETURN
```

O **HOME** da linha 60 foi acrescentado para que o placar pudesse ser mostrado. A linha 110 calcula ao acaso uma posição para [illegible] pato. As linhas 120 e 130 estabelecem condições iniciais para o número de patos (**D**), para o score (**S**) e para o contador de tempo (**CV**).

A linha 210 desenha o pato. A linha 220 verifica se o botão do joystick foi pressionado — neste caso, o valor de um dos endereços dos comandos **PEEK** se torna maior que 127. Caso o programa não funcione, o joystick pode vir [illegible] utilizar o endereço −16285.

A linha 230 verifica se o prazo que o jogador tem para acertar o pato se esgotou, voltando para a linha 200, se isto ainda não aconteceu.

Caso um dos patos não tenha sido abatido dentro do limite de tempo, seu número diminuirá em um. Se ainda sobrarem patos, a sub-rotina 1500 desenhará um deles na tela e o jogo continuará na linha 190.

Se os dez patos já foram desenhados, o jogo terminará [illegible] o score será exibido na porção inferior do vídeo. A linha 260 atualiza o recorde, que é impresso pela linha 270. Uma opção de jogar novamente é então oferecida ao jogador.

A linha 500 foi modificada para incluir um contador de tempo. A sub-rotina 700 desenha ou apaga o pato da tela, conforme a definição de cor atual.

As linhas de 1000 a 1100 verificam se o tiro foi certeiro. As condições do **IF** da linha 1000 permitem uma certa margem de erro ao tiro. Se exigirmos que as coordenadas do pato coincidam com as da mira, o jogo [illegible] tornará muito difícil (lembre-se de que não conseguimos controlar totalmente o cursor em BASIC; além disso, a mira se move de oito em oito pontos).

As linhas 1020 e 1030 produzem uma série de vinte "cliques" no alto-falante, informando o sucesso do tiro. Um novo pato é gerado pela linha 1030. Os pontos ganhos são então calculados, com base no tempo gasto pelo jogador para abater [illegible] ave. O número de patos é diminuído [illegible] o contador de tempo volta a zero.

Se o tiro não atingir o pato, as linhas 1070 e 1080 emitirão três "cliques" e uma multa de cinco pontos será cobrada.

A sub-rotina 1500 apaga o pato de sua antiga posição — linha 1500 — e desenha a mira nesse lugar, calculando ao acaso uma nova posição.

O nível de dificuldade do jogo pode ser alterado, modificando-se [illegible] limite de tempo na linha 230. A contagem de pontos e a multa devem ser alteradas proporcionalmente, nas linhas 1040 e 1090.

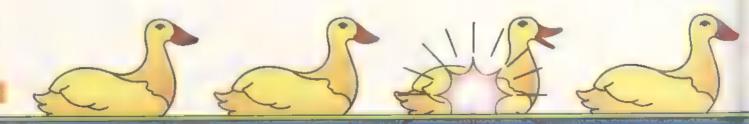
Para o TK-2000, acrescente as linhas válidas para o Apple, com as seguintes modificações:

```
220  IF [illegible] = 46 OR A = 76 THEN
GOSUB 1000: IF D < 1 THEN 250
570  IF A = 46 OR A = 76 THEN
RETURN
```

Essas linhas verificam se um dos botões de tiro do joystick da microdigital foi pressionado. 46 e 76 são os códigos que os botões de tiro enviam ao micro.

T

Ao adicionar as próximas linhas, você terá o jogo completo.

```
80 FOR K=1536 TO 2272 STEP 32
90 READ A,B:POKE K,A:POKE K+1,B
100 NEXT
```

```
110 GET(0,0)-(13,11),D,G
120 GET(0,12)-(13,23),H,G
150 DX=RND(239)+1:DY=RND(178)+1
180 D=10:SC=0
190 TIMER=0
210 PUT(DX,DY)-(DX+13,DY+11),D,PSET
220 IF (PEEK(65280)AND1)=0 GOSUB 2000:IF D<1 THEN 250
230 IF TIMER<200 THEN 200
240 D=D-1:IF D>0 GOSUB 3000:GOTO 190
250 CLS:PRINT @139,"SCORE=";SC
260 IF SC>HI THEN HI=SC
270 PRINT @234,"RECORDE=";HI
280 PRINT @389,"QUER RECOMECAR (S/N)?"
290 A$=INKEY$:IF A$<>"S" AND A$<>"N" THEN 290
300 IF A$="S" THEN 130
310 END
2000 IF PPOINT(X,Y)=1 THEN 2070
2010 PUT(X-6,Y-5)-(X+7,Y+5),H,PSET
2020 PLAY "T5004CEEEG"
2030 GOSUB 3000
2040 SC=SC+250-TIMER
2050 D=D-1:TIMER=0
2060 RETURN
2070 PLAY "T25001DDE"
2080 SC=SC-10
2090 RETURN
3000 PUT(DX,DY)-(DX+13,DY+11),B,PSET
3010 PUT(X-8,Y-5)-(X+9,Y+5),S,PSET
3020 DX=RND(239)+1:DY=RND(178)+1:RETURN
4020 DATA 4,0,25,0,149,0,21,0,5,0,5,64,5,80,21,148,22,148,22,84,21,84,5,80
4030 DATA 48,48,0,0,195,12,51,48,15,192,51,48,204,204,15,192,51,48,192,12,0,0,51,48
```

Dois desenhos estão contidos nas linhas **DATA** 4020 e 4030: o pato e o que restou dele após ser atingido pelo tiro. Os números são lidos e colocados na tela pelas linhas 80 a 100 com auxílio de comandos **POKE**. Dois comandos **GET** são usados para colocar o pato na matriz **D** e o efeito do tiro em **H**.

A linha 150 seleciona ao acaso uma posição para o pato, enquanto a linha 210 utiliza os valores de **DX** e **DY** para desenhá-lo na tela.

Os valores iniciais do número de patos, do score e do contador de tempo são estabelecidos pelas linhas 180 e 190.

Os tiros são detectados pela linha 220. Se o botão da direita for pressionado, o bit 0 do endereço 65280 da memória mudará de 1 para 0; se o botão pressionado for o da esquerda, o bit mudará de 0 para 1. A linha 220 mostra como verificar essas mudanças em 65280. Se utilizarmos os comandos **PEEK** e **AND1** e pressionarmos o botão da direita, teremos o valor zero. **PEEK** e **AND2** verificam o botão da esquerda. Tal como está, a linha 220 se refere ao botão da direita. Se ele for pressionado, o programa será desviado para a subrotina 2000, que verificará se o tiro acertou o alvo.

A parte final da linha 220 averigua se os dez patos já foram desenhados; se a resposta for positiva, o programa irá para a linha 250.

Uma vez pressionado o botão do joystick, a linha 2000 verifica a cor do ponto que está no centro da mira. Se a função **PPOINT** sugerir que a cor é verde, isso significa que o jogador errou o alvo, e o programa irá para a linha 2070, que produzirá um som de erro. Uma multa de dez pontos será cobrada e a sub-rotina terminará na linha 2090.

Se a cor no centro da mira não for verde, isso quer dizer que o tiro atingiu o alvo. Um desenho será colocado sobre o pato, e se ouvirá um som de sucesso. A seguir, é chamada a sub-rotina 3000. Sua função é apagar o que restou do pato e desenhar a mira no mesmo local. A linha 3020 calcula ao acaso a nova posição do pato.

A linha 2040 calcula o score. Este depende do tempo levado para acertar o pato. O número de aves é diminuído de um a cada exemplar abatido e o contador de tempo recomeça de zero.

Após completar a sub-rotina, o programa retorna à linha 230, onde é verificado o prazo para o pato ser atingido. Se a variável **TIMER** for menor que 200, o programa voltará à linha 200. Caso contrário, a linha 240 diminuirá o número de patos, verificando se sobrou algum. Em caso positivo, a sub-rotina 3000 apagará o pato restante e a linha 3020 desenhará um novo pato.

Se os patos tiverem acabado, o score e o recorde serão mostrados pelas linhas 250 a 270. Estas também atualizam o recorde.

As linhas 280 a 310 oferecem ao jogador a possibilidade de jogar novamente.

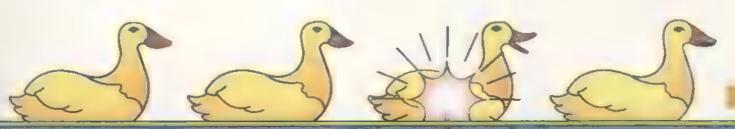
O que fazer para transformar o desenho do pato em um alvo diferente?

A resposta depende do computador que estamos usando. Basicamente, devemos alterar as linhas **DATA** e, talvez, as linhas que colocam caracteres (UDG) na tela.

O programa do Spectrum usa um conjunto de quatro UDG definidos nas linhas 1000 a 1030. Para conjuntos maiores, a linha 180, que coloca o desenho na tela, usando **PRINT**, deve ser modificada.

Usuários do MSX também precisarão mudar as linhas **DATA**. Se usarem mais de um sprite, o tamanho do **FOR** da linha 20 deverá ser alterado e novos **PUT SPRITE**, acrescentados. É possível também que sejam necessárias algumas modificações nos comandos que contêm coordenadas. O programa editor de sprites que fornecemos pode ser útil como ponto de referência.

A chave da modificação das linhas **DATA** do Apple e do TK-2000 está no programa editor de tabelas de figuras. A sub-rotina 700 deve ser alterada em caso de modificação do tamanho da figura. Neste caso, todos os comandos que lidam com coordenadas devem ser igualmente alterados.

Já no TRS-Color é muito simples criar caracteres UDG de qualquer tamanho. Mude as linhas **DATA** e os **FOR...NEXT** que as lêem. Talvez seja necessário mudar também as coordenadas dos comandos **PUT** e **GET**.

CONVERSÕES NO COMPUTADOR

Você tem dificuldade para descobrir quantas polegadas há em um metro ou quantos litros contém um galão? O programa conversor de INPUT resolve [illegible] problema.

Adotado pela maioria dos países do mundo ocidental, o sistema métrico decimal de pesos e medidas tem contribuído, desde o seu surgimento no século XVIII, para facilitar as relações comerciais e o intercâmbio cultural entre os povos. Os Estados Unidos e a Grã-Bretanha, porém, permanecem avessos a ele, preferindo empregar seus próprios sistemas. Essa divergência de critérios tem criado não poucos problemas técnicos, pois grande parte das máquinas e ferramentas em circulação no mercado mundial é produzida nesses dois países, exigindo constantes conversões de medidas, como, por exemplo, de polegadas para centímetros ou, então, de libras para quilogramas.

O programa a seguir faz conversões desse tipo, incluindo até mesmo medidas menos freqüentes, como a de pressão em milímetros de mercúrio (abreviada para mmHG).

Inicialmente, digite o programa e execute-o. Um menu de opções aparecerá imediatamente. A primeira delas (SAIR) permite que você retorne ao BASIC, enquanto as outras se referem aos tipos de unidades que podem ser convertidas. As escolhas possíveis são: comprimento, área, volume, massa (ou peso), pressão e temperatura.

As opções são precedidas por números situados [illegible] sua esquerda. Para selecionar uma delas, pressione [illegible] tecla com o número correspondente. Para converter, por exemplo, unidades de comprimento, pressione a tecla "1". Não importa, por enquanto, [illegible] a conversão será do sistema métrico decimal para o sistema britânico ou vice-versa.

[illegible] UNIDADES

Depois de fazer [illegible] escolha a respeito do gênero de conversão, pressione a tecla com [illegible] número correspondente. O computador mostrará então um segundo menu, que lhe permitirá definir [illegible] unidade [illegible] ser convertida.

Assim, se você pressionou [illegible] tecla "1" para converter unidades de comprimento, tem agora uma lista de todas [illegible] unidades possíveis (polegadas, pés, milímetros, centímetros, metros, etc.). Se quiser converter polegadas em unidades decimais, pressione "1" novamente e digite [illegible] número de [illegible]legadas. Não se esqueça de teclar <ENTER> ou <RETURN> depois disso.

ENCONTRE [illegible] RESPOSTAS

O computador faz as conversões e a seguir mostra os resultados em todas as unidades de um dos sistemas de medidas (métrico decimal ou britânico). Desta forma, se você digitar um valor em polegadas, terá a resposta em milímetros, centímetros, metros e quilômetros.

Depois de mostrar todos os valores, o computador esperará até que uma tecla seja pressionada para prosseguir. Se ela for <ENTER> ou <RETURN>, o programa voltará a mostrar o menu principal. Se você quiser continuar a converter o mesmo tipo de unidade, qualquer outra tecla o levará ao menu anterior.

- CONVERSÃO DO SISTEMA MÉTRICO PARA O SISTEMA BRITÂNICO E VICE-VERSA
- COMO USAR O PROGRAMA
- COMO CONVERTER VALORES DE ÁREA, PESO, COMPRIMENTO, VOLUME, PRESSÃO E TEMPERATURA
- CONVERSÃO DE UNIDADES MISTAS E FRAÇÕES

Para interromper o programa, retorne ao menu principal e tecle "O" para a opção SAIR.

UNIDADES MISTAS

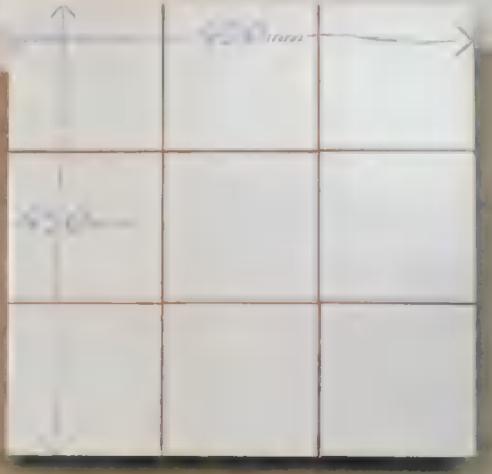
É possível ainda que você precise converter o valor de uma medida britânica em que haja unidades e subunidades. Tomemos, por exemplo, dois pés e seis polegadas. Há duas maneiras de resolver o problema.

O programa aceita números que não sejam inteiros; assim, se você souber quantas polegadas há em um pé, pode calcular a fração decimal correspondente e digitá-la no computador. Neste caso não há dificuldade, visto que seis polegadas são exatamente meio pé. Assim, você digitaria 2.5.

O segundo método (mais empregado no caso de frações de cálculo difícil) consiste em converter o valor em duas etapas. Dessa forma, passe para o sistema decimal, em primeiro lugar, o valor expresso em pés; converta depois o valor em polegadas. Em seguida, tudo o que você tem a fazer é somar os dois resultados (lembre-se de somar sempre valores da mesma unidade decimal).

Se ocorrer o contrário, ou seja, se o número não inteiro for do sistema decimal, calcule a fração conveniente e coloque-a no computador.

TT

```
10 DIML(7),L$(7),A(6),A$(6),V(6
),V$(6),M(5),M$(5),P(4),P$(4)
20 FOR K=0 TO 7:READ L(K),L$(K)
:NEXT
30 DATA 1,POLEGADAS,12,PES,36,J
ARDAS,63360,MILHAS,.03937,MILIM
ETROS,.3937,CENTIMETROS,39.37,M
ETROS,39370,KILOMETROS
40 FOR K=0 TO 6:READ A(K),A$(K)
:NEXT
50 DATA 1,POLEGADAS QUADRADAS,1
44,PES QUADRADOS,6272640,ACRES,
4.0145E9,MILHAS QUADRADAS,.155,
CENTIMETROS QUADRADOS,1550,METR
OS QUADRADOS,1.55E7,HECTARES
60 FOR K=0 TO 6:READ V(K),V$(K)
:NEXT
70 DATA 1,POLEGADAS CUBICAS,172
8,PES CUBICOS,34.67,PINTAS,277.
36,GALOES,.06102,CENTIMETROS CU
BICOS,61.024,LITROS,61024,METRO
S CUBICOS
80 FOR K=0 TO 5:READ M(K),M$(K)
:NEXT
90 DATA 1,ONCAS,16,LIBRAS,35840
,TON. INGLESAS,.03527,GRAMAS,35
.27,KILOGRAMAS,35270,TON. METRI
CAS
100 FOR K=0 TO 4: READ P(K),P$(
K):NEXT
110 DATA 1,PSI,51.73,MM HG,6895
,N/METRO QUADRADO,.0681,ATMOSFE
RAS,68.95,MILIBARES
120 CLS:PRINT"     QUAL CATEGOR
IA (0-6)?"
130 PRINT @72,"0- SAIDA":PRINT
@136,"1- COMPRIMENTO":PRINT @20
0,"2- AREA":PRINT @264,"3- VOLU
ME":PRINT @328,"4- PESO":PRINT
@392,"5- PRESSAO":PRINT @456,"6
- TEMPERATURA"
140 A$=INKEY$:IF A$<"0" OR A$>"
6" THEN 140
150 IF A$="0" THEN CLS:END
160 CLS:ON VAL(A$) GOSUB 1000,1
500,2000,2500,3000,3500
170 GOTO 120
1000 PRINT @9,"comprimento":PRI
NT @34,"SELECIONE UNIDADE ORIGI
NAL":FOR K=0 TO 7:PRINT @136+32
*K,K+1;"- ";L$(K):NEXT
1010 B$=INKEY$:IF B$<"1" OR B$>
"8" THEN 1010
```

```
1020 B=VAL(B$)-1:CLS:PRINT" INT
RODUZA O NUMERO DE ";L$(B)
1030 [illegible] VL
1040 CLS:PRINT VL;L$(B);" EQUIV
ALE A "
1050 IF B>3 THEN 1080
1060 FOR K=0 TO 3:PRINT @96+K*6
4,VL*L(B)/L(K+4),L$(K+4):NEXT
1070 GOTO 1090
1080 FOR K=0 [illegible] 3:PRINT @96+K*6
4,VL*L(B)/L(K),L$(K):NEXT
1090 A$=INKEY$:IF A$="" [illegible] 10
90
1100 IF A$=CHR$(13) THEN RETURN
 ELSE CLS:GOTO 1000
1500 PRINT @13,"area":PRINT @35
,"SELECIONE UNIDADE ORIGINAL":F
OR K=0 TO 6:PRINT @131+32*K,K+1
;"- ";A$(K):NEXT
1510 B$=INKEY$:IF B$<"1" OR B$>
"7" THEN 1510
1520 B=VAL(B$)-1:CLS:PRINT" INT
RODUZA [illegible] NUMERO [illegible] ";A$(B)
1530 INPUT VL
1540 CLS:PRINT VL;A$(B);" EQUIV
ALE A "
1550 IF B>3 THEN 1580
1560 FOR K=0 TO 2:PRINT @96+K*6
4,VL*A(B)/A(K+4),A$(K+4):NEXT
1570 GOTO 1590
1580 FOR K=0 TO 3:PRINT @96+K*6
4,VL*A(B)/A(K),A$(K):NEXT
1590 A$=INKEY$:IF A$="" THEN 15
[illegible]
1600 IF A$=CHR$(13) THEN RETURN
 ELSE CLS:GOTO 1500
2000 PRINT @12,"volume":PRINT [illegible]
35,"SELECIONE UNIDADE ORIGINAL"
:FOR K=0 TO 6:PRINT @133+32*K,K
+1;"- ";V$(K):NEXT
2010 B$=INKEY$:IF B$<"1" [illegible] B$>
"7" THEN 2010
2020 B=VAL(B$)-1:CLS:PRINT" INT
RODUZA O NUMERO DE ";V$(B)
2030 INPUT VL
2040 CLS:PRINT VL;V$(B);" EQUIV
ALE A "
2050 IF B>3 THEN 2080
2060 FOR K=0 TO 2:PRINT @96+K*6
4,VL*V(B)/V(K+4),V$(K+4):NEXT
2070 GOTO 2090
2080 FOR K=0 TO 3:PRINT @96+K*6
4,VL*V(B)/V(K),V$(K):NEXT
2090 A$=INKEY$:IF A$="" THEN 20
90
2100 IF A$=CHR$(13) [illegible] RETURN
 ELSE CLS:GOTO 2000
2500 PRINT @13,"peso":PRINT @35
,"SELECIONE UNIDADE ORIGINAL":F
OR K=0 TO 5:PRINT @136+32*K,K+1
;"- ";M$(K):NEXT
2510 B$=INKEY$:IF B$<"1" OR B$>
"6" THEN 2510
2520 B=VAL(B$)-1:CLS:PRINT" INT
RODUZA O NUMERO DE ";M$(B)
2530 INPUT VL
2540 CLS:PRINT VL;M$(B);" EQUIV
ALE A "
2550 IF B>2 [illegible] 2580
2560 FOR K=0 TO 2:PRINT @96+K*6
4,VL*M(B)/M(K+3),M$(K+3):NEXT
2570 GOTO 2590
2580 [illegible] K=0 TO 2:PRINT @96+K*6
4,VL*M(B)/M(K),[illegible](K):[illegible]
2590 A$=INKEY$:IF A$="" THEN 25
90
2600 IF A$=CHR$(13) THEN RETURN
 [illegible] CLS:GOTO 2500
3000 PRINT @11,"pressao":PRINT
@36,"SELECIONE UNIDADE ORIGINAL
":FOR K=0 TO 4:PRINT @134+32*K,
K+1;"- ";P$(K):NEXT
3010 B$=INKEY$:IF B$<"1" OR B$>
"5" [illegible] 3010
3020 B=VAL(B$)-1:CLS:PRINT" INT
RODUZA [illegible] NUMERO DE ";P$(B)
3030 INPUT VL
3040 CLS:PRINT VL;P$(B);" EQUIV
ALE A "
3050 T=0:FOR K=0 TO 4:IF K=B TH
[illegible] 3070
3060 PRINT @96+T*64,VL*P(K)/P(B
),P$(K):T=T+1
3070 NEXT
3080 A$=INKEY$:IF A$="" THEN 30
8[illegible]
3090 IF A$=CHR$(13) THEN RETURN
 [illegible] CLS:GOTO 3000
3500 CLS:PRINT @10,"temperatura
":PRINT @37,"SELECIONE CONVERSA
O : ":PRINT" CENTIGRADOS PARA F
AHRENHEIT (C) FAHRENHEIT [illegible] C
ENTIGRADOS (F)"
3510 B$=INKEY$:IF B$<>"C" [illegible] [illegible]
$<>"F" THEN 3510
3520 IF B$="C" THEN 3560
3530 PRINT" INTRODUZA GRAUS FAH
RENHEIT":INPUT VL
3540 CLS:PRINT @33,VL;"GRAUS FA
HRENHEIT EQUIVALEM A"
3550 PRINT @97,(VL-32)*5/9;"GRA
US CENTIGRADOS":GOTO 3590
3560 PRINT" INTRODUZA GRAUS CEN
TIGRADOS":INPUT VL
3570 CLS:PRINT @33,VL;"GRAUS CE
NTIGRADOS EQUIVALEM A"
3580 PRINT @97,32+VL*9/5;"GRAUS
 FAHRENHEIT"
3590 A$=INKEY$:IF A$="" THEN 35
90
3600 IF A$=CHR$(13) THEN [illegible]
 ELSE CLS:GOTO 3500
```

```
 5 POKE 23658,8
10 DIM L(8): DIM L$(8,12):
[illegible] A(7): DIM A$(7,16): DIM V
(7): DIM V$(7,16): DIM M(6):
DIM V(7): DIM V$(7,12): DIM M
(6): DIM M$(6,12): DIM P(5):
 DIM P$(5,17)
20 FOR K=1 TO 8: READ L(K),L$
(K): NEXT K
30 DATA 1,"POLEGADAS",12,"PES
",36,"JARDAS",63360,"MILHAS",
.03937,"MILIMETROS",.3937,"CE
NTIMETROS",39.37,"METROS",
39370,"QUILOMETROS"
40 FOR K=1 TO 7: READ A(K),A$
(K): NEXT K
50 DATA 1,"POL.QUADRADAS",144
,"PES QUADRADOS",6272540,"ACR
ES",4.0145E9,"MILHAS QUADRADA
S",.155,"CENT.QUADRADOS",
1550,"METROS QUADRADOS",
 1.55E7,"HECTARES"
 60 FOR K=1 TO 7: [illegible] V(K),V$
(K): NEXT [illegible]
 70 DATA 1,"POL.CUBICAS",1728,
"PES CUBICOS",34.67,"PINTS",
277.36,"GALOES",.06102,"CENT.
CUBICOS",61.024,"LITROS",
61024,"METROS CUBICOS"
 80 FOR K=1 TO 6: READ M(K),M$
(K): NEXT [illegible]
 90 DATA 1,"ONCAS",16,"LIBRAS"
,35840,"TON.INGLESAS",.03527,
"GRAMAS",35.27,"QUILOGRAMAS",
 35270,"TON.METRICAS"
100 FOR K=1 TO 5: READ P(K),P$
(K): NEXT [illegible]
110 DATA 1,"LIBRA/POL.QUADR.",
51.73,"mmHG",6895,"N/METRO QUA
DR.",.0681,"ATMOSFERAS",68.95,
"MILIBARES"
120 CLS : PRINT  INVERSE 1;''
TAB 5;"QUE CATEGORIA (0 A 6)?"
;TAB 31;" "
130 PRINT AT 6,8;"0- SAIDA";AT
8,8;"1- COMPRIMENTO";AT 10,8;"
2- AREA";AT 12,8;"3- VOLUME";
AT 14,8;"4- PESO";AT 16,8;"5-
PRESSAO";AT 18,8;"6- TEMPERATU
RA"
140 LET Z$=INKEY$: IF Z$<"0"
OR Z$>"6" THEN  GOTO 140
 150 IF Z$="0" THEN  CLS :
 STOP
 160 CLS : GOSUB 1000+(VAL Z$-1
 )*500
 170 GOTO 120
1000 PRINT  INVERSE 1;AT 0,12;"
 COMPRIMENTO ": PRINT AT 2,6;"S
ELECIONE UNIDADE ORIGINAL": PRI
NT : FOR K=1 TO 8: PRINT 'TAB 1
0;K;"- ";L$(K): NEXT K
1010 LET B$=INKEY$: IF B$<"1" O
R B$>"8" THEN  GOTO 1010
1020 LET B=VAL B$: INPUT "INTRO
DUZA NUMERO DE ";(L$(B)),VL
1040 CLS : PRINT AT 2,4;VL;" ";
L$(B);" EQUIVALE [illegible] "
1050 IF B>4 THEN  GOTO 1080
1060 FOR K=1 TO 4: PRINT AT K*2
+4,3;VL*L(B)/L(K+4);TAB 18;L$(K
+4): [illegible] [illegible]
1070 GOTO 1090
1080 [illegible] K=1 TO 4: PRINT AT K*2
+4,3;VL*L(B)/L(K);TAB 18;L$(K):
 NEXT K
1090 LET Z$=INKEY$: IF Z$="" TH
EN  GOTO 1090
1100 IF Z$=CHR$ 13 THEN  RETURN
1110 CLS : GOTO 1000
1500 PRINT  INVERSE 1;AT 0,13;"
 AREA ": PRINT AT 2,6;"SELECION
E UNIDADE ORIGINAL": PRINT : FO
[illegible] K=1 TO 7: PRINT 'TAB 10;K;"-
";A$(K): NEXT K
1510 LET B$=INKEY$: IF B$<"1" O
[illegible] B$>"7" THEN  GOTO 1510
1520 LET B=VAL B$: INPUT "INTRO
DUZA O [illegible] [illegible] ";(A$(B)),VL
1540 CLS : PRINT AT 2,4;VL;" ";
A$(B);" EQUIVALE A "
1550 IF B>4 THEN  GOTO 1580
1560 FOR K=1 TO 3: PRINT AT K*2
```

```
MAL
2010  PRINT : HTAB 5: PRINT "ES
COLHA A UNIDADE": PRINT
2020  FOR K = [illegible] TO 6: PRINT : H
TAB 5: PRINT K + 1;"- ";V$(K):
NEXT
2030  GET B$: IF B$ < "1" [illegible] B$
 > "7" THEN 2030
2040 B =  VAL (B$) - 1: HOME :
PRINT "NUMERO [illegible] ";V$(B);"?"
2050  INPUT VL
2060  HOME : PRINT VL; CHR$ (32
);V$(B);" EQUIVALEM A:"
2070  IF B > [illegible] THEN 2100
2080  VTAB 10: FOR K = [illegible] TO 2:
PRINT : PRINT VL * V(B) / V(K +
 4);: HTAB 19: PRINT V$(K + 4):
 NEXT
2090  GOTO 2110
2100  VTAB 10: FOR K = 0 TO 3:
PRINT : PRINT VL * V(B) / V(K);
: HTAB 19: PRINT V$(K): NEXT
2110  GET A$: IF A$ <  >  CHR$
(13) [illegible]  HOME : GOTO 2000
2120  RETURN
2500  INVERSE : PRINT  SPC( 39)
;: HTAB 18: PRINT "PESO": [illegible]
L
2510  PRINT : HTAB 5: PRINT "ES
COLHA A UNIDADE": PRINT
2520  FOR [illegible] = 0 TO 5: PRINT : [illegible]
TAB 5: PRINT [illegible] + 1;"- ";M$(K):
NEXT
2530  GET B$: IF B$ < "1" [illegible] B$
 > "6" [illegible] 2530
2540 [illegible] =  VAL (B$) - 1: HOME :
PRINT "NUMERO [illegible] ";M$(B);"?"
2550  INPUT VL
2560  HOME : PRINT VL; CHR$ (32
);M$(B);" EQUIVALEM A:"
2570  IF B > [illegible] THEN 2600
2580  VTAB 10: FOR K = [illegible] TO 2:
PRINT : PRINT VL * M(B) / M(K +
 3);: HTAB 19: PRINT M$(K + 3):
 NEXT
2590  GOTO 2610
2600  VTAB 10: FOR K = 0 TO 3:
PRINT : PRINT VL * M(B) / M(K);
: HTAB 19: PRINT M$(K): NEXT
2610  GET A$: IF A$ <  >  CHR$
(13) THEN  HOME : GOTO 2500
2620  RETURN
3000  INVERSE : PRINT  SPC( 39)
;: HTAB 16: PRINT "PRESSAO": NO
RMAL
3010  PRINT : [illegible] 5: PRINT "ES
COLHA A UNIDADE": PRINT
3020  FOR K = [illegible] TO 4: PRINT : H
TAB 5: PRINT K + 1;"- ";P$(K):
NEXT
3030  GET B$: IF B$ < "1" OR [illegible]
 > "5" THEN 3030
3040 B =  VAL (B$) - 1: [illegible] :
PRINT "NUMERO DE ";P$(B);"?"
3050  INPUT VL
3060  HOME : PRINT VL; CHR$ (32
);P$(B);" EQUIVALEM [illegible]:"
3070 T = 0: VTAB 10: [illegible] [illegible] = 0
TO 4: IF K = [illegible] THEN 3100
3090  PRINT : PRINT VL * P(K) /
 P(B);: HTAB 19: PRINT P$(K):T
= T + [illegible]
3100  [illegible]
```

```
3110  GET A$: IF A$ <  >  CHR$
(13) THEN  HOME : GOTO 3000
3120  RETURN
3500  INVERSE : PRINT  SPC( 39)
;: HTAB 14: PRINT "TEMPERATURA"
: NORMAL
3510  PRINT : HTAB 5: PRINT "CO
NVERTER:": PRINT
3520  PRINT : HTAB 5: PRINT "CE
NTIGRADO PARA FARENHEIT (C)"
3530  PRINT : HTAB 5: PRINT "FA
RENHEIT PARA CENTIGRADO (F)"
3540  GET B$: IF B$ <  > "C" AN
D B$ <  > "F" THEN 3540
3550  IF B$ = "C" THEN 3590
3560  HOME : INPUT "GRAUS FAREN
HEIT? ";VL
3570  PRINT : PRINT VL;" GRAUS
FARENHEIT EQUIVALEM A ";(VL - 3
2) * 5 / 9;" GRAUS CENTIGRADOS.
": GOTO 3620
3590  HOME : INPUT "GRAUS CENTI
GRADOS? ";VL
3600  PRINT : PRINT VL;" GRAUS
CENTIGRADOS EQUIVALEM A ";32 +
VL * 9 / 5;" GRAUS FARENHEIT."
3620  GET A$: IF A$ <  >  CHR$
(13) THEN  HOME : GOTO 3500
3630  RETURN
```

```
5 DEFSNG A
10 DIM L(7),L$(7),A(6),A$(6),V(
6),V$(6),M(5),M$(5),P(4),P$(4)
20 FORK=0TO7:READ L(K),L$(K):NE
XT
30 DATA 1,polegadas,12,pés,36,j
ardas,63360,milhas,.03937,milím
etros,.3937,centímetros,39.37,m
etros,39370,kilômetros
40 FORK=0TO6:READ A(K),A$(K):NE
XT
50 DATA 1,polegadas quadradas,1
44,pés quadrados,6272640,acres,
4.0145e9,milhas quadradas,.155,
centímetros quadrados,1550,metr
os quadrados,1.55e7,hectares
60 FORK=0TO6:READ V(K),V$(K):NE
XT
70 DATA 1,polegadas cúbicas,172
8,pés cúbicos,34.67,pintas,277.
36,galões,.06102,centímetros cú
bicos,61.024,litros,61024,metro
s cúbicos
80 FORK=0TO5:READ M(K),M$(K):NE
XT
90 DATA 1,onças,16,libras,35840
,tons,.03527,gramas,35.27,kilog
ramas,35270,toneladas
100 FORK=0TO4:READ P(K),P$(K):N
EXT
110 DATA 1,libras/pol ,51.73,mm
Hg,6895,newtons/m ,.0681,atmosf
eras,68.95,milibares
120 CLS:COLOR 15,4,4:PRINT"  Qu
al categoria? (0-6)"
130 LOCATE 5,5:PRINT"0 - Sair":
LOCATE 5,7:PRINT"1 - Compriment
o":LOCATE 5,9:PRINT"2 - Área"
135 LOCATE 5,11:PRINT"3 - Volum
e":LOCATE 5,13:PRINT"4 - Peso":
LOCATE 5,15:PRINT"5 - Pressão":
LOCATE 5,17:PRINT"6 - Temperatu
ra"
140 A$=INKEY$:IFA$<"0"ORA$>"6"T
HEN140
150 IFA$="0"THENCLS:END
160 CLS:ON VAL(A$) GOSUB 1000,1
500,2000,2500,3000,3500
170 GOTO 120
1000 COLOR 15,6,6:PRINT"Comprim
ento":PRINTSTRING$(39,195):LOCA
TE 5,3:PRINT"Selecione a unidad
e:"
1010 FORK=0TO7:LOCATE 5,6+2*K:P
RINTK+1;"- ";L$(K):NEXT
1020 B$=INKEY$:IFB$<"1"ORB$>"8"
THEN1020
1030 B=VAL(B$)-1:CLS:PRINT"Digi
te o número de ";L$(B)
1040 INPUT VL
1050 CLS:PRINTVL;CHR$(32);L$(B)
;" equivalem a:"
1060 IFB>3THEN1090
1070 FORK=0TO3:LOCATE 1,6+2*K:A
=VL*L(B)/L(K+4):PRINTA;TAB(14);
L$(K+4):NEXT
1080 GOTO 1100
1090 FORK=0TO3:LOCATE 1,6+2*K:A
=VL*L(B)/L(K):PRINTA;TAB(14);L$
(K):NEXT
1100 A$=INKEY$:IFA$=""THEN1100
1110 IFA$=CHR$(13)THEN RETURN E
LSE CLS:GOTO1000
1500 COLOR 15,10,10:PRINT"Área"
:PRINTSTRING$(39,195):LOCATE 5,
3:PRINT"Selecione a unidade:"
1510 FORK=0TO6:LOCATE 5,6+2*K:P
RINTK+1;"- ";A$(K):NEXT
1520 B$=INKEY$:IFB$<"1"ORB$>"7"
THEN1520
1530 B=VAL(B$)-1:CLS:PRINT"Digi
te o número de ";A$(B)
1540 INPUT VL
1550 CLS:PRINTVL;CHR$(32);A$(B)
;" equivalem a:"
1560 IFB>3THEN1590
1570 FORK=0TO2:LOCATE 1,6+2*K:A
=VL*A(B)/A(K+4):PRINTA;TAB(14);
A$(K+4):NEXT
1580 GOTO 1600
1590 FORK=0TO3:LOCATE 1,6+2*K:A
=VL*A(B)/A(K):PRINTA;TAB(14);A$
(K):NEXT
1600 A$=INKEY$:IFA$=""THEN1600
1610 IFA$=CHR$(13)THEN RETURN E
LSE CLS:GOTO1500
2000 COLOR 15,12,12:PRINT"Volum
e":PRINTSTRING$(39,195):LOCATE
5,3:PRINT"Selecione a unidade:"
2010 FORK=0TO6:LOCATE 5,6+2*K:P
RINTK+1;"- ";V$(K):NEXT
2020 B$=INKEY$:IFB$<"1"ORB$>"7"
THEN2020
2030 B=VAL(B$)-1:CLS:PRINT"Digi
te o número de ";V$(B)
2040 INPUT VL
2050 CLS:PRINTVL;CHR$(32);V$(B)
;" equivalem a:"
2060 IFB>3THEN2090
2070 FORK=0TO2:LOCATE 1,6+2*K:A
=VL*V(B)/V(K+4):PRINTA;TAB(14);
V$(K+4):NEXT
2080 GOTO 2100
2090 FORK=0TO3:LOCATE 1,6+2*K:A
=VL*V(B)/V(K):PRINTA;TAB(14);V$
(K):NEXT
2100 A$=INKEY$:IFA$=""THEN2100
2110 IFA$=CHR$(13)THEN RETURN E
LSE CLS:GOTO2000
2500 COLOR 15,13,13:PRINT"Peso"
:PRINTSTRING$(39,195):LOCATE 5,
3:PRINT"Selecione a unidade:"
2510 FORK=0TO5:LOCATE 5,6+2*K:P
RINTK+1;"- ";M$(K):NEXT
2520 B$=INKEY$:IFB$<"1"ORB$>"6"
THEN2520
2530 B=VAL(B$)-1:CLS:PRINT"Digi
te o número de ";M$(B)
2540 INPUT VL
2550 CLS:PRINTVL;CHR$(32);M$(B)
;" equivalem a:"
2560 IFB>2THEN2590
2570 FORK=0TO2:LOCATE 1,6+2*K:A
=VL*M(B)/M(K+3):PRINTA;TAB(14);
M$(K+3):NEXT
2580 GOTO 2600
2590 FORK=0TO2:LOCATE 1,6+2*K:A
=VL*M(B)/M(K):PRINTA;TAB(14);M$
(K):NEXT
2600 A$=INKEY$:IFA$=""THEN2600
2610 IFA$=CHR$(13)THEN RETURN E
LSE CLS:GOTO2500
3000 COLOR 15,14,14:PRINT"Press
ão":PRINTSTRING$(39,195):LOCATE
 5,3:PRINT"Selecione a unidade:
"
3010 FORK=0TO4:LOCATE 5,6+2*K:P
RINTK+1;"- ";P$(K):NEXT
3020 B$=INKEY$:IFB$<"1"ORB$>"5"
THEN3020
3030 B=VAL(B$)-1:CLS:PRINT"Digi
te o número de ";P$(B)
3040 INPUT VL
3050 CLS:PRINTVL;CHR$(32);P$(B)
;" equivalem a:"
3060 T=0:FORK=0TO4:IFK=BTHEN308
0
3070 LOCATE 1,6+2*T:A=VL*P(K)/P
(B):PRINTA;TAB(14);P$(K):T=T+1
3080 NEXT
3090 A$=INKEY$:IFA$=""THEN3090
3100 IFA$=CHR$(13)THEN RETURN E
LSE CLS:GOTO3000
3500 COLOR 15,6,6:PRINT"Tempera
tura":PRINTSTRING$(39,195):LOCA
TE 5,3:PRINT"Converter:"
3510 LOCATE 5,6:PRINT"Centígrad
os para Farenheit (C)":LOCATE 5
,8:PRINT"Farenheit para Centígr
ados (F)"
3520 B$=INKEY$:IFB$<>"C"ANDB$<>
"F"THEN3520
3530 IFB$="C"THEN3560
3540 CLS:INPUT"Graus Farenheit"
;VL
3550 LOCATE 3,7:PRINTVL;" graus
 Farenheit equivalem a ":PRINT(
VL-32)*5/9;" graus Centígrados.
":GOTO 3580
3560 CLS:INPUT"Graus Centígrado
s";VL
3570 LOCATE 3,7:PRINTVL;" graus
 Centígrados equivalem a":PRINT
32+VL*9/5;" graus Farenheit."
3580 A$=INKEY$:IF A$=""THEN3580
3590 IFA$=CHR$(13)THEN RETURN E
LSE CLS:GOTO 3500
```

RASTREAMENTO NO SPECTRUM

- O QUE FAZ UM PROGRAMA RASTREADOR
- A BUSCA DOS ERROS
- FUNCIONAMENTO E USOS DO PROGRAMA

Descubra onde se escondem aquelas [illegible] mal escritas. Interrogue o programa marginal que não funciona. Localize erro por erro com [illegible] ajuda do nosso [illegible] rastreador.

É praticamente impossível digitar um programa longo — tal [illegible] [illegible] Assembler — sem cometer alguns erros. Mesmo que tenhamos conferido com todo o cuidado uma listagem, às vezes surgem erros que desafiam até programadores experientes, [illegible] não forem investigados com um recurso especial.

Para prosseguir no curso de linguagem de máquina, [illegible] essencial que seu Assembler funcione. Por isso, antes de mais nada, você deve encontrar todos os erros nele existentes. INPUT traz para os usuários do Spectrum de 48 K de memória um programa rastreador que será muito útil [illegible] busca dos erros eventualmente cometidos na digitação do Assembler. O TRS-Color, [illegible] TRS-80, [illegible] MSX e o Apple possuem programas rastreadores embutidos — são as funções **TRACE, TRACE ON** ou **TRON**.

O programa "TRACE" que apresentamos a seguir vem tanto em Assembly como [illegible] código hexadecimal. Assim, se [illegible] Assembler não está funcionando, você pode usar o monitor da página 92 para entrar os códigos. Caso funcione bem, monte [illegible] programa TRACE com sua ajuda e grave-o em fita, para usá-lo em outros programas que contenham er-

ros. Se você não sabe se o Assembler está funcionando, use a montagem deste programa rastreador como um teste.

COMO USAR O PROGRAMA

Quando um programa BASIC tem um erro, seu micro geralmente informa em que linha está o problema. Isso pode ser suficiente para corrigir um erro em um programa simples e curto. Porém, quando se trata de um programa maior e mais complicado, a mensagem de erro muitas vezes é inútil. Uma linha sempre executada com sucesso pode, de repente, causar um erro porque em outro local do programa uma variável que ela usa assumiu um valor inadequado.

O programa rastreador aqui apresentado simplesmente escreve na tela o número da linha que está sendo executada. Ao contrário das instruções **TRACE** dos demais micros, nosso programa também escreve o código interno do comando BASIC usado no momento.

Para facilitar o uso do programa, convém ter à mão uma listagem do programa, seja ela sua ou de INPUT. Você poderá, assim, acompanhar o programa passo a passo, até atingir o ponto onde ocorreu o erro. Ficará mais fácil também descobrir os valores das variáveis e, ainda, se as condições dos **IF...THEN** estão sendo satisfeitas e se os **GOTO** estão indo para as linhas certas.

FUNCIONAMENTO DO TRACE

Um programa TRACE é um tipo muito especial de rotina em linguagem de máquina, pois funciona simultaneamente a outro programa — o seu, que está em BASIC. Normalmente, não se pode rodar dois programas ao mesmo tempo no computador. O TRACE, na verdade, não constitui uma exceção: ele não roda ao mesmo tempo que o programa em BASIC, mas utiliza um atributo do microprocessador que é a *interrupção*.

Rotinas que empregam esse artifício interrompem o programa principal a cada 20 milésimos de segundo — no Spectrum — e são executadas na fração de segundo que dura essa interrupção. Depois disso, o programa principal continua como se nada tivesse ocorrido, até a interrupção seguinte.

Programas BASIC são sempre interrompidos a cada 0,02 segundo enquanto estão funcionando, para que o computador verifique se alguma tecla foi pressionada. Rotinas que usam interrupções aproveitam-se desse padrão. A interrupção pode não ocorrer, tal como foi explicada, se houver algo conectado na expansão da memória.

Se uma linha BASIC for muito longa, ela poderá ser interrompida mais de uma vez durante sua execução. Nesse caso, o TRACE fornecerá o mesmo número de linha várias vezes. Por outro lado, se uma linha for muito curta — **PRINT** ou **RETURN** isolados, por exemplo — existe uma pequena chance de que TRACE não a detecte. Se isso acontecer, tente provocar uma pausa usando um **FOR...NEXT**.

S

O Spectrum tem dificuldade em imprimir na tela durante uma interrupção. Dois canais diferentes são usados para imprimir na parte superior da tela ou nas duas linhas inferiores, e, se mudarmos de canal enquanto o programa em BASIC está rodando, pode haver complicações.

Para contornar o problema, vamos colocar o número da linha diretamente na tela. Como cada caractere consome oito bytes, simplificamos a parte do programa que cuida disso, fazendo com que os números das linhas sejam sempre colocados no mesmo local da tela, fora da área que o BASIC costuma usar. Ao lado do número, coloca-se o código do comando BASIC que está sendo executado.

O programa TRACE não é chamado de dentro do BASIC, mas por meio de outra rotina em código que, por sua vez, é chamada pelo BASIC. É esta rotina que transfere o controle para o TRACE, durante as interrupções.

Não se esqueça de resguardar o topo da memória — os endereços a partir de 65109 devem estar livres e protegidos por **CLEAR 65109**. Se for usado o Assembler, as origens — ou endereços iniciais — estarão listadas. Se seu Assembler não funcionar, utilize o programa monitor da página 92. O endereço inicial é 65110.

Não se assuste se as traduções das linhas contendo saltos para rótulos que estão adiante da listagem, bem como das linhas que utilizam variáveis e bytes ainda não definidos, saírem erradas na tela. Elas são posteriormente corrigidas pelo próprio Assembler.

Não estranhe também se o Assembler parar a montagem antes de **end**, quando a parte inicial do programa, que contém as linhas **REM** com o programa-fonte, foi muito editada ou teve muitas linhas apagadas. Nesses casos, pode sobrar algum "lixo" no meio da listagem. Este não prejudica a interpretação pelo BASIC, mas confunde o Assembler. Para contornar o problema, redigite a linha imediatamente posterior à última traduzida, e rode o programa novamente. Às vezes é necessário digitar várias linhas de novo.

```
org 65110
ld a,9                   3E 09
ld i,a                   ED 47
im 2                     ED 5E
ret                         C9
org 65120             00 00 00
ld a,62                  3E 3E
ld i,a                   ED 47
im 1                     ED 56
ret                         C9
org 65129                00 00
rst 56                      FF
push af                     F5
ld a,(23622)          3A 46 5C
bit 7,a                  CB 7F
jr z,go                  28 02
pop af                      F1
ret                         C9
go di                       F3
push bc                     C5
push de                     D5
push hl                     E5
push ix                  DD E5
ld de,20726           11 F6 50
ld (posn),de       ED 53 FE FE
ld hl,(23621)         2A 45 5C
call lineno           CD B5 FE
ld de,20731           11 FB 50
ld (posn),de       ED 53 FE FE
ld hl,(23623)         2A 47 5C
ld h,0                   26 00
call statno           CD BB FE
ld hl,23286           21 F6 5A
ld (hl),71               36 47
ld de,23287           11 F7 5A
ld bc,9               01 09 00
ldir                     ED B0
keylp ld a,127           3E 7F
in a,254                 DB FE
or 224                   F6 E0
cp 252                   FE FC
jr z,keylp               28 F6
pop ix                   DD E1
pop hl                      E1
pop de                      D1
pop bc                      C1
pop af                      F1
ei                          FB
ret                         C9
lineno ld bc,-1000    01 18 FC
call prt              CD CE FE
statno ld bc,-100     01 9C FF
call prt              CD CE FE
ld bc,-10             01 F6 FF
call prt              CD CE FE
ld bc,-1              01 FF FF
call prt              CD CE FE
ret                         C9
prt xor a                   AF
prtlp add hl,bc             09
inc a                       3C
jr c,prtlp               38 FC
sbc hl,bc                ED 42
dec a                       3D
add a,48                 C6 30
```

```
push hl                         E5
call print             CD E8 FE
ld hl,posn             21 FE FE
inc (hl)                        34
ld hl,(posn)           2A FE FE
call prtout            CD F5 FE
pop hl                          E1
ret                             C9
print ld bc,(23606) ED 4B 36 5C
ld h,0                     26 00
ld l,a                          6F
add hl,hl                       29
add hl,hl                       29
add hl,hl                       29
add hl,bc                       09
ex de,hl                        EB
ret                             C9
prtout ld b,8              06 08
loop ld a,(de)                  1A
ld (hl),a                       77
inc h                           24
inc de                          13
djnz loop                  10 FA
ret                             C9
posn defw 0                00 00
```

A listagem inclui três programas, e não apenas um. Cada um tem seu próprio endereço inicial. O primeiro, que começa em 65110, ativa o programa principal. O segundo, que começa em 65120, desliga o programa TRACE. O terceiro, que começa em 65129, faz o rastreamento.

As instruções **ld a, 9** e **ld i,a** colocam o número 9 dentro do registro I. Não há uma instrução que carregue o registro I diretamente com um número. O 9 é interpretado como o byte mais significativo de um vetor de interrupção que possui dois bytes. O byte menos significativo é fornecido pelo computador,

sendo geralmente igual ■ 255. Assim, o vetor de interrupção aponta para o endereço 9*256 + 255, que ■ igual a 2559. O valor contido nos endereços 2559 e 2560 é 65129, ■ endereço inicial do nosso programa TRACE.

O processo pode parecer confuso, ■■ rotinas com interrupções devem ser endereçadas indiretamente. Note que o endereço 2559 fica na ROM. Se colocássemos no registro I um número maior que 64, para apontarmos para um endereço da RAM, onde poderíamos colocar o número que quiséssemos — 65129 já estava lá —, veríamos que os caracteres se desmanchariam no vídeo.

O mnemônico **im 2** altera o modo de interrupção. A roti■■ que desliga o programa principal coloca o valor 62 no registro I; 62 era o conteúdo original desse registro. *im 1* restabelece o modo de interrupção normal.

O programa principal começa com ■ instrução **rts 56**. Ela determina que o microprocessador execute sua rotina normal de interrupção — fazendo a varredura do teclado e atualizando o relógio do sistema.

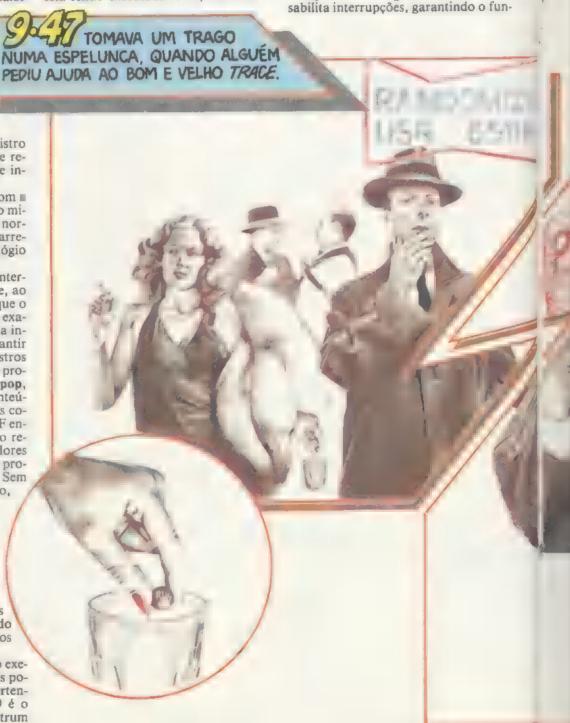
Ao programar uma rotina com interrupção, deve-se levar em conta que, ao final de sua execução, é essencial que o conteúdo de todos os registros seja exatamente igual ao que havia antes da interrupção. A única maneira de garantir isso é colocar o conteúdo dos registros ■■ pilha, usando **push**, no início do programa, e retirá-lo de lá usando **pop**, quando o programa acabar. Os conteúdos de BC, DE, HL e IX são todos colocados na pilha. O conteúdo de AF entra antes porque o acumulador e o registro F — que contêm os sinalizadores — são utilizados para verificar se ■ programa BASIC está sendo rodado. Sem um programa BASIC funcionando, não há nada ■ ser rastreado.

Para verificar se existe um programa em curso, coloca-se o conteúdo da posição 23622 no acumulador. As posições 23621 e 23622 contêm o número da linha BASIC que está sendo executada. Embora os números das linhas BASIC sejam armazenados em formato alto-baixo na área do programa, aqui eles são armazenados no formato baixo-alto.

Se nenhum programa está sendo executado, o número contido nas duas posições será muito elevado para pertencer ■ uma linha BASIC — 9999 é o maior número de linha que o Spectrum aceita. Assim, o conteúdo de 23622 (o byte mais significativo) é colocado no acumulador e a instrução **bit 7,a** examina o bit mais significativo do conteúdo de A. Se esse bit for 1, o programa não está sendo executado. Uma instrução ■■ verifica o valor de determinado bit de uma posição ou registro. Por exemplo, **bit 4,a** testa ■ quarto bit do acumulador.

Se o bit mais significativo do byte mais significativo da linha BASIC que está sendo executada for 0, o sinalizador zero é estabelecido e **jr z,go** faz com que o programa vá para ■ próxima ocorrência do rótulo **go**. Se o bit 7 do acumulador for 1, o sinalizador não é estabelecido e o saldo não ocorre. O programa continua na próxima instrução, que recupera o valor de AF da pilha e retorna ao interpretador BASIC.

Uma vez que haja um programa sendo rodado e que ele tenha sido transferido para o rótulo **go**, ■ instrução **di** desabilita interrupções, garantindo o fun-

cionamento contínuo de nossa rotina de interrupção.

A instrução **ld de,20726** coloca no registro DE o endereço da posição da tela imediatamente anterior àquela onde colocaremos um caractere. Como o programa usará muito esse endereço, ele é colocado ■ variável **posn**, por meio do comando **ld posn,de.** Nosso Assembler interpreta a variável **posn** como uma posição de memória que tem um nome. Como nenhuma instrução permite colocar um número diretamente dentro de uma variável ou posição de memória, usamos um registro como intermediário.

Coloca-se no par HL o conteúdo das posições 23621 ■ 23622, que é ■ número da linha BASIC que está sendo executada no momento. Podemos então chamar a sub-rotina de rótulo **lineno**.

O número da linha BASIC em HL está em hex. Para convertê-lo para decimal, começamos por calcular o algarismo dos milhares: subtraímos 1000 do número em HL repetidamente, contando quantas vezes foi possível fazer tal operação. Colocamos, assim, o valor −1000 no par BC ■ chamamos ■ sub-rotina **prt**, que faz ■ subtrações. Nessa sub-rotina, a primeira coisa que o microprocessador faz é **xor** ■ — isto é, um ou exclusivo. Em linguagem de máquina, um ■ sempre age no registro A; assim, ■ ■ faz um ou exclusivo do acumulador com ele mesmo, ■ que sempre resulta em 0. Esta é uma maneira rápida de colocar 0 no acumulador, já que **ld a,0** tem um byte ■ mais.

add hl,bc subtrai 1000 do conteúdo de HL. Somar −1000 ■ mais rápido que subtrair 1000, já que dispensa cuidar do sinalizador *carry* ("vai um"). O acumulador é, então, incrementado e atua como contador. **jr c,prtlp** verifica o sinalizador carry e salta quando ele é estabelecido.

Se considerarmos o conteúdo de BC entenderemos como tudo funciona. Vimos no artigo da página 142 que os números negativos são representados dentro do computador por números positivos muito grandes, mas de comprimento limitado. O par de registros BC contém dezesseis números ■ binários, menos 1000 em decimal. Se qualquer número maior que 1000 for somado a BC, o valor do resultado ultrapassará a capacidade do par HL e o sinalizador carry será estabelecido.

Quando isso ocorre, ■ um salto, 1000 é novamente subtraído de HL, o contador é incrementado e ■ processo recomeça. Ele se repete até que o conteúdo de HL seja menor que 1000; ■ adição com BC, então, não estabelece o sinalizador carry. Neste ponto, porém, ■ subtração já foi feita e a incrementação do contador ocorreu uma vez mais que o necessário. Assim, somamos 1000 a HL — na realidade subtraímos −1000 — e decrementamos o acumulador.

O acumulador contém, agora, o algarismo dos milhares. Adicionando 48, obtemos o código ASCII do caractere. O conteúdo de HL é reservado na pilha para quando formos calcular as centenas, dezenas e unidades. Em seguida, chamamos a sub-rotina **print**.

A primeira instrução dessa sub-rotina é **ld bc, (23606)**, que coloca o conteúdo das posições 23606 ■ 23607 em BC. Essas posições contêm a variável do sistema que aponta para o endereço inicial do conjunto de caracteres da ROM. O conteúdo de H ■ reduzido a zero e o do acumulador é transferido para L, que conterá, assim, ■ código ASCII do caractere que queremos escrever. O registro HL é usado, então, como um acumulador de 16 bits.

Para colocar na tela o caractere desejado, temos que obter seu formato dentro do conjunto de caracteres que fi■ na ROM. Como cada caractere tem oito bytes, o endereço inicial dos oito bytes que fornecem o padrão do caractere é oito vezes o código ASCII do caractere, mais o endereço inicial do conjunto de caracteres.

Em vez de multiplicar ■ código ASCII por oito, é mais fácil duplicá-lo — somando-o consigo mesmo — três vezes. Observe que a operação não é feita no registro A porque o valor certamente ultrapassará oito bits — 48 X ■ = 384, posição inicial do caractere 0, ■ maior que 255, máxima capacidade de um registro de oito bits. Essa conta, porém, não ultrapassará os dezesseis bits.

Para obter o endereço do primeiro byte do caractere desejado, adiciona-se o resultado da operação ao conteúdo de BC. Os conteúdos de HL ■ DE são intercambiados para que HL possa ser usado novamente.

ret determina ■ retorno do microprocessador da sub-rotina ■ **ld hl, posn** coloca ■ posição da tela contida em **posn** no registro HL. A instrução indireta **inc (hl)** incrementa o conteúdo de **posn**. A sub-rotina que coloca o caractere na tela — **prtout** — é, então, chamada.

A primeira instrução dessa sub-rotina é **ld b, 8**, que coloca 8 no registro B. Ele será usado como contador para os oito bytes do caractere que serão transferidos para ■ tela. O acumulador é carregado com o conteúdo da memória cujo endereço está em DE, ou seja, com o primeiro byte do caractere. O conteúdo do acumulador é transferido para a posição de memória cujo endereço está em HL (o local apropriado da tela). Exceto em um comando de ação em bloco, não se pode transferir ■ conteúdo de uma posição de memória diretamente para outra sem utilizar um registro como intermediário. Não há uma instrução do tipo **ld (hl), (de)**, por exemplo. Essa transferência requer, portanto, duas instruções, ■ o que ela realmente faz é colocar a primeira linha de oito pontos do caractere na tela.

O conteúdo do registro H é, então, incrementado. Como H contém o byte mais significativo do par HL, o conteúdo desse par aumenta em 256, de forma ■ corresponder ■ endereço da próxima linha do caractere.

Incrementando o conteúdo de DE, obtemos o endereço da próxima linha do caractere, dentro do conjunto de caracteres, e **djnz loop** faz ■ microprocessador voltar à sub-rotina **prtout**, para colocar na tela os outros bytes do caractere. Este laço é repetido oito vezes, incrementando H a cada volta, para colocar na tela o próximo byte abaixo do anterior, ■ incrementando DE, para obter o padrão dos oitos pontos a serem colocados na tela, do conjunto de caracteres da ROM. Ao mesmo tempo, a instrução **djnz** (decrementa saltando ■ não for zero) decrementa ■ registro B. Assim, quando ■ processo estiver ■ oitava repetição — e os oito bytes que compõem o caractere tiverem sido colocados na tela —, o caractere correspondente ao algarismo dos milhares estará ■ vídeo, o registro B conterá zero, a condição não-zero do comando **djnz** não será satisfeita ■ o microprocessador passará à próxima instrução, que é **ret**, retornando para onde foi chamada a sub-rotina.

O comando **pop hl** recupera os dois últimos bytes colocados na pilha. Se olharmos para trás, veremos que se trata do resto que ficou em HL após ■ cálculo do algarismo dos milhares. **ret** manda o microprocessador de volta à linha onde está **statno ld bc, −100** e o processo ■ repete para calcular ■ algarismo das centenas. Quando este for colocado na tela ao lado do algarismo dos milhares, o algarismo das dezenas — e, depois, o das unidades — ■ calculado da ■ maneira ■ colocado ■ tela na posição adequada.

Depois disso, ■ microprocessador retorna para onde ■ sub-rotina **lineno** foi chamada. O valor 23731 é colocado em **posn**, um endereço da tela um pouco ■ direita de onde foi colocado o número da linha. Obtém-se, assim, espaço suficiente para quatro dígitos do número e um espaço em branco.

O conteúdo das posições 23623 e 23624 ■ colocado em HL; estas posições contêm o número da instrução BASIC que está sendo executada no momento. O valor máximo que o código de uma instrução BASIC pode assumir é 128; o byte mais significativo é, portanto, desnecessário e deve ser reduzido ■ zero pela instrução **ld hl, 0**. A sub-rotina que cuida da conversão do número para decimal, bem como de sua impressão na tela, é executada novamente, só que desta vez é chamada por meio do rótulo **statno**, já que são necessários apenas três dígitos.

Quando ■ código da instrução BASIC é novamente impresso ■ tela, o microprocessador volta ao programa principal, onde executa ■ pequena sub-rotina que coloca no vídeo ■ caracteres invertidos. Como a rotina leva menos de 20 milésimos de segundo, não há a ■ chance de percebermos que um caractere foi desenhado normalmente e depois invertido.

ld hl,23286 ■ o endereço da posição anterior ao primeiro dígito na tabela de atributos, que fornece ■ "moldura" ao número. O número 71 — que produz fundo preto e caracteres brancos — é colocado nesta posição. O endereço da próxima posição é colocado em DE ■ 9, no par BC, que será usado como contador.

A instrução de armazenamento em bloco **ldir** coloca ■ conteúdo da posição de memória dada por HL na posição dada por DE, decrementa B ■ verifica se o valor de B foi reduzido a zero. Se não foi, o processo é repetido. Em outras palavras, ■ instrução copia os atributos fundo preto e caracteres brancos do primeiro caractere nos outros nove caracteres adjacentes.

PAUSA NO PROGRAMA

As nove instruções seguintes permitem parar temporariamente o programa TRACE — e, também, ■ programa BASIC — ■ qualquer momento. A instrução **in** recebe informações vindas de uma das portas de entrada. No nosso caso, estamos interessados no teclado, que envia dados ■ microprocessador pela porta 254. Queremos verificar ■ alguma tecla da porção inferior direita do teclado — de B até **BREAK/SPACE** — foi pressionada. Assim, **ld a,127** coloca 127 dentro do acumulador para ser usado como parâmetro da instrução **in**. Juntos, os comandos **ld a,127** e **in a, 254** colocam ■ valor enviado pela porção citada do teclado no acumulador.

De B a **BREAK/SPACE** existem apenas cinco teclas, cada qual representa-

da por um bit dentro do número enviado ao microprocessador. Sobram, portanto, três bits. Se executarmos ■ operação lógica ■ entre o conteúdo do acumulador e 224, faremos com que os três bits mais significativos se tornem 1 — 224 ■ 11100000 ■ binário.

Quando uma tecla não está sendo pressionada, seu bit dentro do valor enviado é 1. Quando ela ■ pressionada, este valor muda para zero. Para provocar uma pausa no TRACE, deve-se pressionar duas teclas simultaneamente: <**SIMBOL SHIFT**> e <**BREAK/SPACE**>. Com a pausa, o teclado pode ser usado normalmente para editar linhas, mesmo que o programa esteja "ligado".

Quando pressionamos <**BREAK/SPACE**>, o bit zero do número enviado passa de 1 para 0. <**SIMBOL SHIFT**> faz ■ mesmo ao bit um. Se não pressionarmos nenhuma das duas teclas e cada um dos três bits mais altos for transformado em um, ■ número enviado pela porção do teclado em questão será 255, ou 11111111. Mas, se pressionarmos as duas teclas ao mesmo tempo, o número enviado será 252, ou 11111100.

Assim, quando o número enviado pela porta 254 entra no acumulador, é comparado com 252 pela instrução **cp 252**. Se o acumulador contiver 252, ■ sinalizador zero será estabelecido. Então, **jr z** (salto relativo se zero) faz com que a rotina seja repetida. Enquanto as duas teclas permanecerem pressionadas, as repetições prosseguirão. Como o nosso programa interrompeu o programa principal, este também não prosseguirá enquanto o microprocessador não terminar ■ execução da rotina.

Se nenhuma tecla é pressionada, o processador passa às instruções **pop ix**, **pop hl**, **pop de**, **pop bc** ■ **pop af**, que recuperam o conteúdo original dos registros anteriores ■ interrupção.

A instrução **ei** habilita ■ ocorrência de interrupções e **ret** faz o microprocessador voltar ao interpretador BASIC, que continuará ■ execução do programa principal.

A última instrução da listagem Assembly, **defw posn**, não terá códigos correspondentes no programa-objeto que ■ Assembler vai criar. Esse tipo de instrução é usado pelo Assembler apenas para reservar dois bytes, onde serão colocados números utilizados pelo programa. No nosso caso, ■ que fica armazenado ali é ■ valor de **posn**.

A instrução **defw** (definir dois bytes) reserva um espaço para colocar dados, permitindo também que se dê um nome ■ ele — **posn**, por exemplo. Outra instrução semelhante é **defb**, que "define um byte".

COMO ENCONTRAR OS ERROS

Para usar o TRACE, carregue o programa BASIC a ser corrigido. Em seguida, reserve o topo da memória com **CLEAR 65109** e carregue o TRACE.

Antes de rodar o programa rastreador, grave-o em fita. Se o seu Assembler estiver funcionando, faça-o pela via usual. Assim, você gravará ■ programa-fonte, das linhas **REM**, junto com o resto do Assembler. Quando o programa rastreador estiver funcionando, é melhor gravá-lo em fita, já montado em linguagem de máquina. Para isso, digite:

```
SAVE "TRACE" CODE 65110,176
```

Para carregá-lo, digite primeiro **CLEAR 65109** e, em seguida:

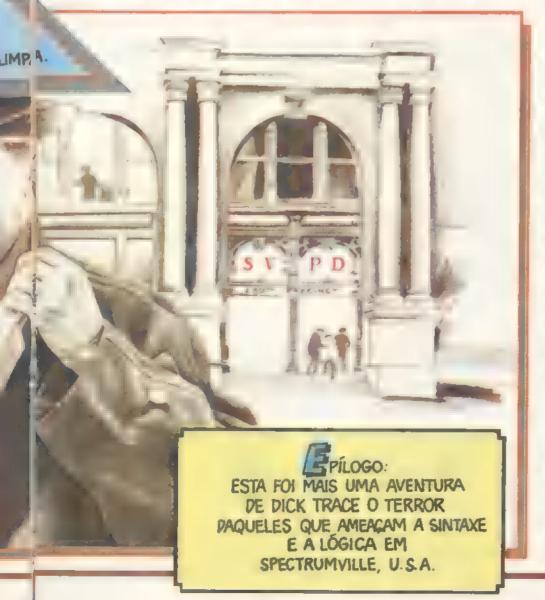
```
LOAD "" CODE
```

Se ■ Assembler não estiver funcionando, use a opção de gravação do seu monitor. Para ligar o TRACE, digite:

```
RANDOM USR 65110
```

Nada acontecerá até que um programa em BASIC seja executado, com **RUN**. Depois disso, aparecerão no vídeo o número da linha e o código do comando que estiver sendo executado.

O problema mais fácil de se detectar com o auxílio do programa rastreador é o "laço sem fim". Se você observar que os mesmos números de linha se repetem mais e mais vezes, pode estar certo de que há algo errado em algum **GOTO**. Por outro lado, ■ certas linhas nunca forem executadas, é provável que algum **IF...THEN** contenha condições erradas. Caso queira observar os eventos mais detalhadamente, use a pausa.

Para desligar o programa rastreador, digite:

```
■■■■ USR 65120
```

ARTE GRÁFICA EM SEU MICRO

Conhecendo mais a fundo os comandos gráficos de [illegible] micro, você poderá fazer melhor uso dos [illegible] disponíveis. [illegible] mais: saberá contornar todas [illegible] [illegible] limitações.

Já reunimos muita informação a respeito dos comandos gráficos. Ainda assim, é possível que não estejamos fazendo o melhor uso deles.

Os comandos de cor, por exemplo, são muito mais versáteis do que parecem [illegible] primeira vista e fazem muito mais do que simplesmente pintar blocos. Aqui estão algumas idéias e técnicas que talvez o ajudem a aprimorar seus gráficos.

S

Os comandos **PLOT** e **DRAW** do Spectrum podem ser usados de várias maneiras, mas sabemos que nem sempre produzem o efeito desejado. Muitas vezes, isto se deve às limitações da tela de alta resolução gráfica, que não aceita cores diferentes em pontos adjacentes dentro de um mesmo quadrado na tela de texto. Mas há outros fatores que interferem nos resultados. O melhor [illegible] fazer é tentar aproveitar [illegible] máximo o efeito obtido, seja ele qual for.

Por exemplo, se o método de sombreamento utilizado apresenta um resultado pouco uniforme, elabore o programa de modo que o efeito pareça proposital. [illegible] se algumas cores estão se sobrepondo em certas áreas do trabalho, ou [illegible] você precisa de mais de duas cores num mesmo quadrado, projete o desenho de modo que qualquer mudança de cor aconteça num novo quadrado.

Restarão ainda os problemas com as curvas mas, com um certo esforço, será possível diminuir seus efeitos.

O importante é que, diante da impossibilidade de obter um determinado efeito, você pelo menos esteja preparado para adaptar e usar todos [illegible] recursos disponíveis. Daí, sim, na medida em que aprendermos mais sobre a máquina, procuraremos desenvolver novos métodos de refinamento do desenho.

COMO SOMBREAR A TELA

Sabemos que o Spectrum dispõe de eficientes recursos para a elaboração de desenhos (veja artigos das páginas 113 [illegible] 232, por exemplo).

Mostraremos agora como sofisticar ainda mais [illegible] gráficos, explorando melhor os comandos de desenho e de cor que já conhecemos. E por que não aumentar nosso repertório de figuras ou aprender a utilizá-las como base para outras?

Este pequeno programa desenha pontos em posições aleatórias. Observe quanto tempo leva para preencher a tela toda.

```
10 LET x=INT (RND*256)
20 LET y=INT (RND*176)
30 PLOT x,y
40 GOTO 10
```

Como você viu, demora bastante, o que torna o método pouco aconselhável para quem quer preencher completamente uma grande área num gráfico. Entretanto, o programa é muito útil quando se pretende sombrear uma área — e isto ele faz com certa rapidez.

Com [illegible] simples acréscimo de uma linha criamos um efeito ainda melhor. Adicione a seguinte linha a seu programa e rode-o novamente:

```
25 IF (X>35 [illegible] X<90) [illegible] (Y>
   35 AND Y<90) [illegible] GOTO 10
```

Logo se observa que [illegible] computador vai deixando um quadrado limpo de pontos — ou seja, vazio. A linha 25 verifica se [illegible] valores de x [illegible] y estão entre 35 e 90; o computador salta, em seguida, para [illegible] linha 10, para gerar novos valores. Obtemos, então, uma área vazia entre 35 e 90, em ambas as direções; forma-se, assim, um quadrado nessas coordenadas.

Poderíamos utilizar [illegible] técnica num jogo: por exemplo, depois de escrever algo num quadrado, o resto da tela seria preenchido gradualmente, enquanto as palavras continuariam visíveis. A mesma técnica também será útil para destacar uma mensagem na tela, como o título de um programa ou jogo.

- ADAPTAÇÃO DOS RECURSOS DISPONÍVEIS
- USE MELHOR AS CORES
- COMO SOMBREAR E COLORIR A TELA DO SPECTRUM
- MAIS SOBRE O CIRCLE E O [illegible] NO MSX
- PSET, PRESET E COLOR NO TRS-COLOR
- O FLASH NO APPLE

As condições na linha 25 parecem ser complicadas. No artigo da página 34, vimos como usar **AND** e [illegible] para incluir mais de uma condição num **IF...THEN**. O significado dessas funções coincide exatamente com a tradução dos dois termos — ou seja, E [illegible] OU, respectivamente. Na linha 25, portanto, x tem que ser maior que 35 **AND** (E) menor que 90, **AND** y tem que ser menor que 90 **AND** maior que 35; só assim [illegible] computador executará o **GOTO** 10.

Os parênteses entre [illegible] duas metades da linha separam as condições para x [illegible] y (que na verdade são as mesmas, mas devem ser separadas por um **AND**). No [illegible] exemplo, poderíamos retirar os parênteses, pois utilizamos somente **AND**. Caso usássemos tanto **AND** quanto **OR**, os parênteses seriam essenciais. Veremos [illegible] que o [illegible] faz no exemplo seguinte. Podemos mudar a área que pretendemos deixar intacta da maneira que quisermos — experimente, por exemplo, substituir o **AND** entre parênteses por um **OR**. Em vez de quadrado vazio, obteremos [illegible] cruz.

Mude a linha 25 para:

```
25 IF (X>110 [illegible] X<145 [illegible] Y>40
   [illegible] Y<136) [illegible] (X>40 [illegible] X<
   215 AND Y>70 AND Y<100) THEN
   GOTO 10
```

Você pode deixar qualquer área vazia, mas não se esqueça de que, quanto mais complicada for esta área, mais condições terá o **IF...THEN**, e mais tempo o computador levará para verificá-las. Mesmo um triângulo — ou um círculo — requer muitas condições e, assim, o computador demorará para desenhar [illegible] pontos. Aconselhamos, no caso de [illegible]breamento, restringir as áreas a linhas horizontais e verticais, pois elas levam menos tempo para serem verificadas e, portanto, aparecem logo na tela.

DESENHO E COR

Em artigos anteriores vimos como desenhar no Spectrum figuras bastante detalhadas. Os principiantes em programação gráfica, porém, deparam-se com um problema: de um modo geral, essa máquina apresenta uma grande limitação quando se trata de desenhar uma figura [illegible] sombreá-la com cor. Caso não saiba do que estamos falando, veja este exemplo:

```
 5 [illegible] 7: PAPER 7: INK 4:
 CLS
10 FOR x=1 TO 10
20 PRINT " ■■■■■■■■■■ "
30 NEXT [illegible]
40 PLOT  INK 2;0,175
50 DRAW  INK 2;120,-80
60 INK 1: CIRCLE 100,95,50
```

O programa preenche [illegible] área em verde, usando blocos gráficos, depois traça uma linha diagonal vermelha e um círculo azul — pelo menos, deveria ser assim. Na prática, porém, todos os quadrados por onde [illegible] linha passa mudam de cor. É claro que existem maneiras de se evitar que isso ocorra, mas [illegible] maioria delas precisa de longos programas para que o Spectrum manipule as informações.

Mais adiante, em outro artigo, veremos como fazê-lo. Por enquanto, ficaremos com uma alternativa simples: evitar as áreas problemáticas. Foi [illegible] que fizemos nos programas de gráficos anteriores, como o do campo de golfe, apresentado na página 233. Nesse caso, escolhemos primeiro a cor da tela e, depois, desenhamos as linhas, observando que nenhuma delas passasse sobre áreas já ocupadas por caracteres gráficos.

Para que você entenda melhor e, ao mesmo tempo, experimente um trabalho com arcos, digite [illegible] programa a seguir. Ele desenha um carro colorido.

```
 80 BORDER 2: PAPER 6: INK 0:
 CLS
 90 FOR n=8 TO 15
100 CIRCLE 80,47,n: CIRCLE 180
,47,n
110 NEXT n
120 PLOT 62,51: [illegible] 38,-5,-PI
130 DRAW 60,0: DRAW 40,0,-PI
200 FOR n=1 TO 10: [illegible] a,b,c:
[illegible] a,b,c: NEXT n
210 PLOT 112,48: DRAW 45,0:
DRAW 22,20,-PI/2
220 DRAW 7,15: [illegible] -35,23:
[illegible] -40,0: DRAW 0,-58
230 PLOT 115,104: [illegible] 34,0:
DRAW 30,-19
240 [illegible] -64,-5,-.25: DRAW 0,
24
250 PLOT 40,83: DRAW 52,5,.3:
DRAW 6,20,.3
260 PLOT 30,55: DRAW -5,1:
DRAW 0,5: DRAW 5,1
270 PLOT 240,56: DRAW 5,1:
DRAW 0,5: [illegible] -5,1
280 PLOT 36,75: DRAW [illegible] 2;
184,0
290 PRINT  OVER 1; INK 2;AT 13
,4;"▣":AT 13,28;"▣"
300 FOR n=31 TO [illegible] STEP -1:
PLOT 0,n: DRAW  INK 4;255,0:
NEXT n
500 DATA 40,8,.2,0,10,0,-30,15
,.2,-20,5,.2
510 DATA -40,25,0,-60,-2,.1,
-50,-25,.2
520 DATA -10,-20,-.25,0,-8,0,
31,-3,.2
```

As linhas 90 a 110 desenham uma série de círculos, cada um maior que o anterior, formando as rodas do carro. O Spectrum não consegue traçar círculos perfeitos numa área quadrada. Em con-

seqüência das diferenças de curvatura, alguns pontos se perdem, ou melhor, não são incluídos em curva alguma. Como esses pontos não são desenhados, o pneu do carro adquire ■ aparência pontilhada. Poderíamos encarar tal efeito como um problema, mas não há razão para isso, já que, na verdade, a roda ganhou um visual ainda mais realístico. Como afirmamos, para se desenhar bem é necessário fazer bom uso das características da máquina, e não se atrapalhar com elas.

A linha 120 desenha um ponto na coordenada 62,51, que é ■ posição inicial das linhas que formam o carro. Usando o laço **FOR...NEXT** na linha 200, e as linhas 120 e 130, o Spectrum lê (**READ**) as informações necessárias para desenhar (**DRAW**) o contorno do carro. Os dados (**DATA**) para ■ laço **FOR...NEXT** estão nas linhas 500 ■ 520. Note que todas as curvas estão na forma x, y, z, que ■ o comando para desenhar uma linha curva, ou seja, um arco, cuja curvatura é especificada por z.

Os primeiros dois números são os mesmos de sempre: definem o quanto acima e o quanto à direita do último ponto queremos o próximo ponto. O terceiro número especifica ■ ângulo da curva. O comando que desenha ■ capô curvo do carro ■ **DRAW 38, −5 − PI**. Tente mudar ■ último número (PI ■ mais ou menos 3,14) ■ veja a diferença de curvatura do capô.

Os detalhes internos ao contorno constituem seções do programa: a porta e seu vidro estão nas linhas 210 a 240; o vidro traseiro, na linha 250; os pára-choques, nas linhas 260 a 270 e a lista na linha 280. As linhas 290 ■ 300 simplesmente dão um toque colorido ■ desenho: a linha 290 coloca as lanternas (vermelhas) e a linha 300 faz ■ grama sob o carro.

Posicionando cuidadosamente o carro e ■ grama, evitaremos o problema de ter duas cores num mesmo quadrado. Observe, ■ programa acima, ■ grama sob o carro. Este foi posicionado com suas rodas chegando até o fundo de um quadrado, de modo que o ponto seguinte (onde começa a grama) fica em outro quadrado, não interferindo, assim, na cor dos pneus.

Como se vê, ■ necessário planejar em detalhes ■ posição de cada elemento. Mas ■ sempre isso é possível: no nosso caso, por exemplo, ■ listra vermelha altera a cor de algumas partes do carro. O importante, porém, é evitar alterações de cor ■ lugares onde elas apareceriam muito.

Para praticar com o **DRAW**, aconselhamos que tente mudar algumas partes do carro; somente assim será possível saber exatamente o que cada linha faz e, ■ mesmo tempo, familiarizar-se com o comando. O ideal seria podermos imaginar a figura acabada ■ precisar desenhá-la desde ■ começo.

O USO DO FLASH

O **FLASH**, um comando de utilização muito simples, é ideal para destacar mensagens na tela, principalmente em jogos. Ele não usa parâmetro nenhum e, em geral, vem antes do comando **PRINT**. Não se esqueça, porém, de que o **FLASH** é um comando separado do **PRINT**: por essa razão, deve vir numa outra linha ■ separado por dois pontos. Exemplo:

```
FLASH:PRINT" PISCA-PISCA "
```

Quando o computador encontra ■ comando **FLASH** em uma linha de programa, ele faz "piscar" tudo o que for mandado para a tela a partir daquela linha, ou seja, ■ conteúdo de todos os **PRINT** que vierem depois de **FLASH**. O **FLASH** mostra alternadamente o inverso ■ o normal dos caracteres exibidos ■ tela.

Digite e rode o seguinte programa:

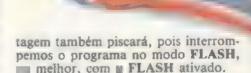
```
10  HOME : VTAB 10
20  FLASH
30  PRINT "ISTO ESTA PISCANDO"
40  FOR T = ■ TO 2000: NEXT
50  HOME : VTAB 10
60  PRINT "   ISTO  TAMBEM   "
70  FOR T = 0 TO 2000: NEXT
80  GOTO 10
```

Tecle <CTRL-C> para parar e liste o programa. Tudo continua piscando, pois o **FLASH** ainda está ativado. Para desativá-lo, digite **NORMAL** ■ aperte a tecla <**RETURN**>. Liste novamente.

O comando **NORMAL** desativa o **FLASH** e pode ser usado tanto dentro de programas como fora deles ou, em outras palavras, tanto ■ modo direto como ■ indireto. Substitua as linhas 60, 70 e 80 e adicione a linha 90. As modificações são as seguintes:

```
60  NORMAL
70  PRINT "ISTO NAO ESTA"
80  ■ T = 0 TO 2000: NEXT
90  GOTO 10
```

Se apertarmos <CTRL-C> no "ISTO NÃO ESTÁ" e depois listarmos o programa, veremos que a listagem não pisca, pois interrompemos ■ programa quando este estava em modo **NORMAL**. Mas, se apertarmos <CTRL-C> no "ISTO ESTÁ PISCANDO", a lis-

tagem também piscará, pois interrompemos o programa no modo **FLASH**, ■ melhor, com ■ **FLASH** ativado.

O comando **INVERSE** mostra os caracteres em modo inverso, ou seja, letras pretas num fundo branco.

O programa que se segue faz a apresentação dos três comandos em seus respectivos modos.

```
10   ■ : VTAB 10
20   FLASH
30   PRINT : PRINT "  FLASH  "
40   GOSUB 110
50   INVERSE
60   PRINT : PRINT " INVERSE "
70   GOSUB 110
■    NORMAL
90   PRINT : PRINT "  NORMAL"
100   ■
110   FOR T = 0 TO 2000: NEXT
120   RETURN
```

O programa que apresentamos a seguir emprega gráficos já vistos ■ poderia ser usado para ensinar inglês ■ crianças.

O programa desenha figuras para as três primeiras letras do alfabeto. Para a letra A, desenha uma "apple" (maçã); para a letra B, uma "ball" (bola) e, para a C, um "cat" (gato). A ampliação do programa, ou seja, ■ desenho de figuras para ■ outras letras, ficará ■ cargo de cada um. A figura relacionada à letra D poderia ser, por exemplo, um "dog" (cão) e estaria situada a partir da linha 4000. Para dizer ao computador aonde ir caso a letra escolhida seja ■ D, basta acrescentar uma vírgula ■ 4000, depois do 3000 na linha 60. A figura da letra ■ poderia ser colocada a partir da linha 5000 e assim por diante.

Se quiser aperfeiçoar o programa, escreva também o nome da figura e a letra escolhida, como mostram as ilustrações da bola e do gato. Para isso, utili■ a sub-rotina que desenha letras em tela de alta resolução, já apresentada mais

de uma vez em INPUT (por exemplo, no artigo da página 354). Deixamos ■ adaptação desta sub-rotina como um desafio a programadores mais experientes.

```
5 SCREEN 0:COLOR 1,15,15
10 PI=4*ATN(1):CLS:LOCATE 9,10
20 PRINT"ME ■ ■ LETRA => ?"
30 A$=INKEY$:IFA$<"A"ORA$>"Z"TH
EN30
40 LOCATE 28,10:PRINTA$
50 FORT=1TO500:NEXT
■ A=ASC(A$)-64:ON A GOTO 1000,
2000,3000
70 GOTO 30
1000 SCREEN2:COLOR15,11,12:CLS
1010 CIRCLE(128,96),40,6,,,1
1020 PAINT(128,96),6
1030 FOR Z=1 TO 50 STEP .3
1040 PSET((100+Z*.5)+RND(1)*25,
80+RND(1)*40),15
1050 NEXT
1060 CIRCLE(90,70),40,12,0,PI/2
,1
1070 CIRCLE(112,50),10,12,,,.5
1080 CIRCLE(140,55),10,12,,,.5
1090 PAINT(112,50),12
1100 PAINT(140,55),12
1110 GOTO 3280
2000 SCREEN2:COLOR15,1,9:CLS
2010 C1=INT(RND(1)*14)+1
2020 C2=INT(RND(1)*14)+1
2030 CIRCLE(128,96),10,C1,,,.7
2040 CIRCLE(128,96),50,C1,,,.7
2050 PAINT(140,90),C1
2060 CIRCLE(128,96),30,C2,,,.7
2070 CIRCLE(128,96),10,C2,,,.7
2080 PAINT(140,90),C2
2090 GOTO 3280
3000 SCREEN2:COLOR1,3,4:CLS
3010 REM CORPO/CABEÇA/OLHOS
3020 CIRCLE(128,106),40,1,,,1.5
3030 CIRCLE(128,46),20,1,,,1.15
3040 CIRCLE(121,40),5,1,,,.5
3050 CIRCLE(135,40),5,1,,,.5
3060 PAINT(128,46),1
3070 PAINT(128,106),1
3080 ■ ORELHAS
3090 X1=108:X2=113:X3=103
3100 LINE(X1,15)-(X2,28),9
3110 LINE(X1+40,15)-(X2+40,28),
9
3120 LINE(X2,28)-(X1,41),9
3130 LINE(X2+40,28)-(X1+40,41),
■
3140 LINE(X1,41)-(X3,28),9
3150 LINE(X1+40,41)-(X3+40,28),
9
3160 LINE(X3,28)-(X1,15),9
3170 LINE(X3+40,28)-(X1+40,15),
■
3180 PAINT(105,28),9:PAINT(150,
28),9
3190 REM CAUDA
3200 CIRCLE(75,106),35,1,1.2*PI
,1.8*PI,1
3210 CIRCLE(75,106),33,15,1.2*P
I,1.8*PI,1
3220 CIRCLE(128,50),1,15,,,1
3230 REM BIGODE
3240 LINE(128,52)-(150,55),15
3250 LINE(128,52)-(150,53),15
3260 LINE(128,52)-(106,55),15
3270 LINE(128,52)-(106,53),15
3280 FOR T=0 TO 2000:NEXT:GOTO 5
```

Logo ■ início, o programa pede por uma letra. A linha 60 é responsável pelo cálculo do valor da letra, que será guardado na variável **A**, ■ também pela escolha da sub-rotina relacionada ■ letra escolhida. No nosso programa, se a letra escolhida foi A, B ■ C o programa saltará para a linha 1000, 2000 ■ 3000, respectivamente. Na linha 60, não há opção para as letras maiores que C; para estas letras, o programa volta para a linha 30.

A sub-rotina que desenha ■ maçã começa na linha 1000. As linhas 1030 ■ 1050 desenham pontos brancos em posições aleatórias, para dar um visual mais realista ao desenho. A linha 1060 desenha um arco que vai de 0 ■ PI/2 graus, representando o pequeno caule.

A bola começa na linha 2000; C1 ■ C2 são cores aleatórias de números 1 a 15. Suponhamos que a bola esteja dividida em duas: ■ bola maior, que será pintada na cor C1, ■ ■ bola menor (dentro da maior), que receberá a cor C2. Pintamos ■ bola maior primeiro porque, no MSX, a cor com que colorimos a figura deve ser a mesma do seu contorno. Seria impossível, por exemplo, fazer em branco o círculo que divide as duas bolas e, depois, pintar a maior de azul e ■ menor de verde. Nenhuma cor utilizada para colorir aceitará outra cor como delimitação de contorno. Nossa saída foi desenhar na cor C1 a bola maior ■ pintá-la nessa cor, antes de desenhar e, também, pintar na cor **C2** a bola menor. Observe que foi preciso redesenhar o círculo menor na cor C2 (linha 2070) para que a bola menor o aceitasse como parte dela. Omita ■ linha 2070 para ver o que acontece.

A sub-rotina que desenha ■ gato começa na linha 3000. As linhas 3110, 3130, 3150 e 3170 desenham a orelha esquerda, que está 40 pontos à direita da outra — por isso somamos 40 às mesmas coordenadas ■ da outra orelha. As linhas que desenham ■ cauda — 3200 e 3210 — seguem o mesmo princípio do desenho do caule, na linha 1060. A linha 3220 desenha o nariz.

Já vimos como usar ■ **PSET** no desenho de círculos ■ curvas seno ■ cosseno. Vamos agora explorar um pouco mais ■ funcionamento ■ o de seus relacionados **PRESET**, **PCLS** e **COLOR**.

O COMANDO PSET

Podemos dizer que ■ **PSET** "acende", ■ cor escolhida, a menor unida-

de gráfica em qualquer **PMODE**.

No [illegible] **4**, somente um ponto é aceso; nos **PMODE** 2 [illegible] 3, acendem-se dois pontos ao mesmo tempo, e nos **PMODE** 0 e 1, quatro. Não confunda [illegible] **PSET** empregado dessa maneira com o **PSET** usado anteriormente no comando **LINE** (veja [illegible] artigo da página 113).

Rode este programa:

```
10 PMODE 0,1
20 PCLS
30 SCREEN 1,1
40 FOR X=0 TO 255
50 Z=(X-127)/10
60 Y=95-150*Z/(1+Z*Z)
70 IF Y<0 [illegible] Y>191 THEN 90
80 PSET (X,Y,5)
90 NEXT
100 GOTO 100
```

O programa desenha um gráfico no modo de duas cores, mas poderia também desenhá-lo em **PMODE** 4 ou **PMODE** 2. Este é o gráfico de uma função matemática obscura [illegible] foi escolhido porque produz [illegible] traço interessante.

Ao usar **PSET** devemos dizer à máquina onde queremos acender o ponto (ou conjunto de pontos) e em que cor. As cores disponíveis dependem do **PMODE** utilizado ou do conjunto de [illegible] escolhido.

A cor da tela será a que tiver menor número, entre as cores disponíveis no conjunto escolhido, [illegible] não ser que queiramos mudá-la.

Como veremos mais tarde, quando usarmos **PSET** [illegible] modo de duas cores precisaremos especificar a cor de maior número; caso contrário, não veremos nada. No modo de quatro cores, a situação é diferente, já que podemos escolher qualquer uma das três cores de maior número.

Retornando ao programa: [illegible] conjunto de cores preto e amarelo foi escolhido na linha 30. As linhas 40, 50 e 60 calculam valores para [illegible] e [illegible] [illegible] [illegible] linha 80 acende o ponto na coordenada x, y. O último argumento é [illegible] cor que o ponto receberá — no [illegible] caso, 5 (amarelo).

MODO DE QUATRO CORES

Tente mudar a linha 10 para:

```
10 PMODE1,1
```

Agora, rode o programa novamente. Não se preocupe [illegible] nada acontecer — é isto o que se espera. Na verdade, estamos acendendo pontos em amarelo num fundo também amarelo. Temos que mudar o último argumento do **PSET**, na linha 80, para selecionar uma cor diferente. Neste conjunto de cores, podemos escolher entre ciano (6), magenta (7) e laranja (8). Tente mudar o argumento 5, na linha 80, para 6, 7 ou 8. Se quiser, mude o conjunto de cores na linha 30 para:

```
30 SCREEN1,0
```

Agora, podemos usar cores de números 1 [illegible] 4, embora 1 acenda o ponto na mesma cor do fundo.

Só como curiosidade: [illegible] escolhermos o conjunto de cores 0 e usarmos 5, 6, 7 ou 8 como último argumento do **PSET**, a máquina não enviará mensagem de erro, pois automaticamente subtrai 4 dos números.

O COMANDO PRESET

O **PRESET** é o inverso do **PSET**, ou seja, ele "apaga" [illegible] ponto especificado nos argumentos entre parênteses. Pode-se também imaginar o **PRESET** como acendendo um ponto, ou conjunto de pontos, [illegible] mesma cor do fundo.

Para observar o funcionamento do **PRESET**, rode o programa novamente e, então, mude a linha 80 para:

```
80 PRESET(X,Y)
```

Mas preste atenção: não use [illegible] comando **RUN** para rodar o programa com a linha alterada, pois se o fizer [illegible] tela será limpa.

Para verificar como o **PRESET** apaga ponto por ponto, digite **GOTO** 30 e, em seguida, tecle <**RETURN**>.

A COR DA TELA

É possível mudar ■ cor da tela para qualquer uma das cores do conjunto escolhido. Para isso, basta acrescentar o número da cor junto ao **PCLS**, ■ linha 20.

Supondo que a linha 30 seja **SCREEN1,1**, tente as seguintes mudanças na linha 20: **PCLS6**, **PCLS7** e **PCLS8**. **PCLS5** tem o mesmo efeito de **PCLS**, pois ambos limpam a tela na cor amarela. Terminadas as mudanças, faça ■ linha 20 **PCLS** novamente.

FUNDO E PRIMEIRO PLANO

Como vimos no artigo da página 113, o comando **LINE** usa **PSET** e **PRESET** de uma maneira diferente da que acabamos de ver. Quando empregado com o comando **LINE**, ■ **PSET** diz ao computador para desenhar na cor do primeiro plano e o **PRESET** diz que o desenho deve ser feito ■ cor do fundo.

Quando ligamos ■ TRS-Color, a cor do primeiro plano é a de maior número do conjunto de cores (veja seu manual para ■ números das cores) e a cor de fundo é a de menor número.

Mas suponha que estamos num modo de quatro cores e queremos traçar uma linha em uma das outras cores do conjunto. Não teremos problema, pois existe um comando em BASIC que nos permite escolher as cores do fundo e do primeiro plano.

Digite este programa para ver como funciona o comando **COLOR**:

```
10 PMODE 3,1
20 PCLS
30 SCREEN 1,0
40 FOR K=1 TO 4
50 FOR J=1 TO 4
60 COLOR K,J
70 LINE(0,50*K+10*J-55)-(255,50
*K+10*J-55),PSET
■ LINE (0,50*K+10*J-52)-(255,5
0*K+10*J-52),PRESET
90 ■ J,K
100 GOTO 100
```

O programa desenha pares de linhas paralelas, uma na cor de fundo e outra na cor de plano. Não ■ pode ver todas ■ linhas, pois o fundo ou o plano ajustados em verde coincidem com ■ cor da tela.

O comando **COLOR** na linha 60 produz a variação das cores. Observe que os comandos **LINE** não são alterados durante o programa, exceto ■ coordenadas finais.

O primeiro número depois do comando **COLOR** é a cor do plano e o segundo, a cor do fundo. Nada impede que a cor do fundo seja a mesma do plano, embora não haja muita utilidade nisso.

■ usarmos somente **PCLS** no programa, ■ cores de fundo e tela serão iguais, mas, se especificarmos um número junto ao **PCLS**, ■ cores podem ser diferentes. Da mesma maneira, a cor do plano nem sempre ■ a mesma cor com que desenhamos. O único comando gráfico que faz uso das cores de plano ■ fundo é o **LINE** e, como vimos, ainda assim podemos desenhar tanto ■ cor de fundo como na de plano.

CRIE SUA PRÓPRIA AVENTURA

Agora que você já viu como programar uma aventura, ■ hora ■ partir para criações originais. Eis como usar o jogo ■ INPUT como ■ para elaborar o seu próprio programa.

A esta altura, você deve ter um jogo completo de aventura gravado em fita. Durante seu desenvolvimento, vimos como reunir no programa os elementos que o compõem. Agora, você aprenderá a utilizar o jogo já pronto como base para elaborar suas próprias aventuras.

Algumas dicas de como proceder para fazer seu jogo foram dadas ■ longo dos últimos artigos. Todos os detalhes serão tratados aqui em maior profundidade.

Nem sempre poderemos ser muito específicos sobre as alterações necessárias, uma vez que elas dependerão muito das características da sua aventura; mas você poderá fazê-las com facilidade, se seguir as instruções. Algumas técnicas parecerão confusas no início, mas, começando com uma aventura simples e curta, você logo ■ sentirá à vontade para programar jogos mais complexos. Não tente fazer muitas alterações de uma só vez; vá devagar, estudando as seções deste artigo e fazendo as alterações uma a uma.

Comandos BASIC que não foram bem compreendidos podem se tornar mais claros com uma consulta à seção de *programação BASIC* correspondente — um grande número de comandos já foi abordado por essa seção de INPUT.

Os usuários do TK-2000 e do Spectrum de 16k não poderão estender muito sua aventura, o que não significa que não possam aumentá-la um pouco ou acrescentar outros locais. Sempre é possível, por exemplo, trocar alguma característica da aventura por um novo local; tudo dependerá do que se considerar mais importante.

A ESCOLHA DO TEMA

Antes de escrever ■ aventura, será preciso escolher uma trama ou história.

A estrutura de uma aventura bem-sucedida ■ geral não foge ■ certos padrões — há começo, meio e fim, ordenados conforme ■ seqüência em que os problemas devem ser resolvidos. Mas não é necessário ser Agatha Christie para programar aventuras. Existem mui■ fontes de idéias, como você já deve ter percebido; em todo caso, aqui vão algumas sugestões.

Podemos basear nossa aventura em ■ assassinato. O jogo começaria, por exemplo, ■ sala, onde ■ corpo esfaqueado jaz sem vida sobre o tapete ensangüentado. O objetivo do aventureiro seria descobrir o assassino. Substituiríamos, então, o fiscal da Receita por um tipo mal-encarado — ou pelo mordomo. Ele poderia ajudar ou atrapalhar o jogador com pistas verdadeiras ou falsas.

Se você quiser manter ■ tema da "caça ao tesouro", encontrará várias maneiras de utilizá-lo. Recorra, por exemplo, ao clichê tradicional, com piratas e tudo, ou transforme o jogador em sobrevivente de um desastre aéreo que vitimou todos os membros de uma expedição. Mandar o jogador rumo ao futuro, por outro lado, pode ser uma boa idéia. Procurando novos mundos, ele acaba preso em um planeta hostil, anos-luz distante da civilização, por causa de um defeito em sua nave espacial. O jogador teria como objetivo achar um jeito de voltar à Terra. Os locais poderiam ser cheios de perigo e haveria bastante espaço para incluir — ou ocultar — diversas rotas de fuga.

Esta idéia de fuga sugere outros te■ de aventuras, em que o objetivo do jogador seria, por exemplo, escapar de lugares como Alcatraz, San Quentin ou o Presídio da Ilha Grande. Use livros e jornais como fontes de inspiração. Talvez você até encontre um mapa do lugar real para fazer o mapa da aventura!

Casos de espionagem também podem dar aventuras interessantes, assim como fatos históricos. As cruzadas, por exemplo, podem

se revelar um cenário muito interessante, assim como qualquer outra guerra ou batalha.

E, como sugestão final, um tema da moda: o aventureiro sobrevive ao holocausto nuclear. Os elementos para o jogo são quase infinitos: mutantes, busca de proteção contra a radiatividade, ameaça de gangues de bandidos famintos, procura de água e comida não contaminadas e assim por diante.

NOVOS LOCAIS

A aventura de INPUT é bem pequena; as suas logo serão bem maiores que ela.

Se seguirmos as instruções das páginas 226 a 231, teremos um mapa que poderá ser introduzido no computador. O mapa da nossa aventura tem 24 locais (6x4) e apenas 12 deles são utilizados. Podemos aproveitar o programa original — o que dá menos trabalho, mas exige mais habilidade — ou escrever um novo — o que envolve mais esforço, porém menos confusão. Se você escolher a primeira alternativa, deverá carregar o programa original da fita ou listá-lo através da impressora. As modificações dependerão do tama-

nho do novo mapa. Se ele for menor ou igual ■ antigo, conserve a numeração original. Caso contrário, será preciso fazer um novo desenho, numerando-o segundo os mesmos princípios.

TI TK-2000

Pronto o mapa, você deverá fazer novas descrições dos locais. Elas vão substituir as antigas, que ficavam a partir da linha 5000, no programa.

Após cada descrição, informe as possíveis saídas do local, usando as variáveis **N**, **S**, **E** e **W**, que correspondem a norte, sul, leste e oeste. Elas podem conter ■ valores 1 e 0. 0 significa que não há saída naquela direção, enquanto 1 quer dizer ■ contrário. O esforço de digitar linhas **REM** indicando os locais pode valer a pena.

Em seguida, modifique as linhas 330 ■ 350 que contêm **ON...GOSUB**. O primeiro número que se segue ao **GOSUB**, ■ linha 330, se refere à linha onde o computador vai encontrar ■ descrição do local 1. Se não houver local 1 na sua aventura (não é preciso incluir todas as posições do mapa), 0 é usado. O próximo número se refere à descrição do local 2, ■ assim por diante. Cada posição do mapa deve ter seu número.

MOVIMENTO

Se o tamanho do mapa for diferente do original, você precisará alterar ■ linhas 1000 ■ 1040 da rotina de movimento. Mais especificamente, as linhas que se referem às direções norte e sul — 1010 e 1030 — devem ser modificadas se o mapa não tiver seis posições de largura. Isto porque, para seguir nestas direções, subtraímos ou adicionamos 6 ■ número do local. A nova largura do mapa deve substituir o 6.

OBJETOS

Grandes modificações deverão ser feitas nas linhas 160 ■ 260, pois os objetos certamente serão diferentes.

O número de objetos da nova aventura dá o valor da variável NB; deve ser o primeiro elemento na linha **DATA** 200. Ele terá a função de dimensionar as matrizes na linha 180 e de determinar o tamanho de laços **FOR...NEXT** em vários pontos do programa.

Se usarmos uma linha **DATA** para cada objeto, o programa ficará mais claro. No entanto, se o número de objetos for muito grande, pode-se optar por colocar mais de um por linha. Em todo caso, a ordem deve ■ precisa, pois cada grupo de três dados contém elementos de três diferentes matrizes. A ordem é: número do local onde se encontra o objeto, descrição curta e descrição longa. Se o objeto só aparece mais tarde na aventura — após ser encontrado, por exemplo, como o olho — ou se surge ao acaso, como ■ fiscal da Receita, o número do local deve ser zero.

NOVAS PALAVRAS

É preciso preparar uma lista de todas as palavras permitidas ao jogador. Ela deve incluir comandos simples, como "AJUDAR" e "LISTAR", bem como comandos com duas palavras, tais como "PEGAR LÂMPADA" e "MA-

TAR SENHORIO".

Os comandos de duas palavras dividem-se em **V$** e **N$** — verbo e substantivo, respectivamente —, embora nem sempre de acordo com a definição gramatical. Vamos considerar todos os comandos simples e todas as primeiras palavras dos comandos compostos como sendo verbos. Eles devem ser agrupados de acordo com seu significado — "MASTIGAR" e "COMER", ou "CHEIRAR" e "FAREJAR", por exemplo, ficarão juntos. Cada um desses grupos terá um número — anote-os, pois será difícil decorar a que grupo de palavras correspondem.

Agora vamos alterar o programa. A rotina que trata dos verbos vai da linha 110 a 150. Os verbos devem ser colocados — cada qual seguido de seu número — nas linhas **DATA** 140 e 150.

Não se esqueça de redimensionar as matrizes, na linha 130, de acordo com o número total de verbos utilizados.

ROTINAS QUE ENTENDEM OS VERBOS

Cada categoria — ou número — de verbo vai precisar de uma rotina diferente.

É difícil dar instruções específicas a respeito dessas rotinas, porque um verbo usado em uma aventura pode não servir para nada em outra.

Contudo, algumas rotinas — como "PEGAR" e "LARGAR" — podem ser úteis em qualquer aventura. Você não precisará modificá-las, a menos que o roteiro de seu jogo seja muito inovador. Da mesma forma, não será necessário alterar a rotina "LISTAR", nas linhas 1070 a 1130, caso os nomes das matrizes e a variável **NB** (número de objetos) permaneçam os mesmos.

Uma rotina que sempre permite adaptações é a da lâmpada, já que situações que envolvem acendê-la são muito comuns em aventuras.

As rotinas restantes, de um modo geral, não podem ser aproveitadas. Você precisará, assim, escrever novas rotinas. Ao fazê-lo, lembre-se de que uma das finalidades é verificar se o jogador escolheu corretamente o objeto e o local para executar aquilo que o verbo indica. Se o local não é adequado, o programa deve orientá-lo com mensagens do tipo "AQUI NÃO". Seja qual for o resultado da ordem dada, o jogador precisa ser informado — isto é, qualquer palavra que entre no computador deve provocar uma mensagem.

Depois de planejar as rotinas de execução das ordens verbais, coloque-as no programa. Se seu programa tem uma numeração parecida com a da aventura de INPUT, o lugar dessas rotinas é entre as linhas 1070 e 2999.

O computador selecionará a rotina correspondente ao verbo usado pelo jogador. A linha 510 é a responsável por isso, e deverá ser modificada. Para tanto, consulte os números de sua lista de verbos. Coloque, então, na frente do **ON...GOTO**, o endereço inicial das rotinas, na mesma ordem numérica da lista dos verbos correspondentes.

ROTINA DE AUXÍLIO

A última rotina a ser modificada é a que trata de "AJUDAR" o jogador. Verifique onde o auxílio é necessário e use **IF...THEN** para dar uma mãozinha ao aventureiro.

Outras características da aventura original devem ser modificadas ou eliminadas, conforme as exigências do novo jogo. É o caso, por exemplo, da rotina que faz o coletor de impostos aparecer. Há também a alternativa de começar o jogo em uma posição diferente; para isso, basta alterar a linha 280.

LISTA DE VARIÁVEIS

Para que você entenda melhor o funcionamento do programa original, fizemos uma lista das variáveis e matrizes por ele utilizadas.

R$() matriz contendo os verbos e respostas do jogador.
R() matriz de números de verbos.

Elementos correspondentes das duas matrizes acima referem-se ao mesmo verbo.

OB() matriz com o número do local onde se encontra o objeto.
OB$() matriz com a descrição curta do objeto.
SI$() matriz com a descrição longa do objeto.

Elementos correspondentes das três matrizes acima referem-se ao mesmo objeto.

NB número de objetos da aventura, usado para dimensionar matrizes e tamanhos de laços **FOR...NEXT**.
L local onde o jogador se encontra no momento.
LA sinalizador do estado da lâmpada; contém 1, se a lâmpada está acesa, e 0, se está apagada.
TA sinalizador do coletor de impostos.
N,S,E,W saídas de um local; contém 1, se há saída naquela direção, e 0, se não há.
I$ entrada ou resposta completa do jogador, antes de ser dividida em verbo e substantivo.
V$** verbo em **I$.
N$** substantivo em **I$.
I contém um número correspondente a um verbo; usado para selecionar a rotina de execução adequada.
IN número de objetos que estão sendo carregados pelo jogador.
A$ resposta à pergunta "Quer jogar novamente?"
G contém o número do objeto deixado de lado pelo jogador — que é o elemento **G** na matriz **OB**.

O funcionamento do programa do Spectrum é um pouco diferente, devido à ausência de comandos **ON...GOSUB** e **ON...GOTO** nesta máquina. Não há desvantagem nisso. O programa funciona muito bem e sua modificação não apresenta maiores dificuldades.

DESCRIÇÕES DOS LOCAIS

O primeiro passo para adaptar o programa do Spectrum é incluir as descrições dos locais de seu novo mapa.

Elas são colocadas no final do programa, da mesma forma que na aventura original. Devem permanecer em ordem e seguidas da linha que contém as saídas — variáveis **N**, **S**, **E** e **W**, correspondendo a norte, sul, leste e oeste, acompanhadas dos números 1 ou 0, indicando se há ou não saída naquela direção. Não se esqueça das linhas **REM**, para saber onde está cada descrição na listagem.

NOVAS PALAVRAS

Faça uma lista de todas as palavras que o jogador poderá usar durante o jogo. Neste momento, interessam particularmente os verbos — a primeira palavra em um par, ou palavras usadas isoladamente e que nem sempre são verbos no sentido gramatical. Reúna as palavras que têm o mesmo sentido (e, portanto, o mesmo efeito no desenrolar da aventura), como "PEGAR" e "APANHAR" ou "MATAR" e "ATIRAR". Cada grupo precisará de um número. Inclua-os também na lista.

Os verbos seguidos de seus números devem ser colocados nas linhas **DATA** 140 e 150. Não se esqueça de deixá-los entre aspas.

Deveremos ainda redimensionar as matrizes linha 120 e ajustar tamanho do laço **FOR...NEXT**, na linha 130, de acordo com o número de palavras utilizadas.

ROTINAS QUE ENTENDEM OS VERBOS

Cada categoria de verbos precisará de uma rotina.

É difícil dar instruções específicas sobre elas, uma vez que quase sempre servem apenas ao jogo de origem. Todavia, algumas das rotinas da aventura de INPUT podem ser utilizadas, pois são comuns quase todos os jogos desse tipo.

É o caso de "PEGAR" e "LARGAR": fundamentais para o uso dos objetos, serão inteiramente aproveitadas. A rotina que executa a ordem "LISTAR" também é aproveitável, servindo à maioria dos jogos. Sua aventura ficará bem estranha sem uma função desse tipo.

As demais rotinas dependem das exigências do seu jogo. Para ter uma orientação, observe como programamos as rotinas do nosso jogo. Preste atenção nos locais e nos objetos sobre os quais um determinado verbo pode ter ação.

As rotinas devem ser capazes de verificar se o local e os objetos da ação são apropriados. Devem também estar preparadas para informar os resultados ao jogador, imprimindo mensagens de erro quando ele pedir coisas proibidas.

Uma vez escritas, as rotinas ocuparão o espaço entre as linhas 1390 e 2999, inclusive. O computador selecionará essas rotinas de acordo com o verbo usado pelo jogador.

A MATRIZ G

O Spectrum guarda as descrições dos locais as linhas das rotinas dos verbos dentro da matriz **G**.

O próximo passo consiste, assim, em colocar os números das linhas iniciais das rotinas em **G**. As linhas 40 a 70 contêm números. As primeiras três linhas referem-se às descrições dos locais. A última contém os números das linhas iniciais das rotinas dos verbos.

As dimensões da matriz **G** dependem do número de linhas **DATA** e de quantos números linha **DATA** mais longa contém. O primeiro índice no comando **DIM G (M,N)**, ou seja, **N**, corresponde ao tamanho da linha **DATA** mais longa. O segundo, **M**, à quantidade destas linhas.

Coloque todos os dados nas linhas de 40 a 70 — se eles forem muitos, use números de linha intermediários. Verifique quantos números estão incluídos na linha mais longa. Acrescente às linhas mais curtas tantos zeros quantos forem necessários para que fiquem do mesmo tamanho. Não se esqueça das vírgulas.

Se os números estiverem corretamente colocados em **G**, as linhas 330 a 350 vão selecionar a descrição adequada local onde se encontra jogador. O segundo índice da matriz corresponde à linha onde se encontra o número. Verifique se os índices estão corretos em cada comando **GOTO** de seu novo programa, especialmente se você aumentou o número de linhas **DATA**.

MOVIMENTO

Se o tamanho do mapa de sua aventura for diferente, linhas 1000 a 1040 deverão ser modificadas, pois tratam do movimento do jogador.

Mais especificamente, as linhas que tratam de movimentos para o norte e para o sul — 1010 e 1030 — precisam ser alteradas se seu mapa não tiver seis locais de largura. A nova largura deve substituir número 6.

OS OBJETOS

Grandes alterações deverão ser feitas nas linhas 160 a 260, uma vez que seus objetos serão diferentes dos da nossa aventura.

Anote o número de objetos o comprimento das maiores descrições (tanto das curtas quanto das longas). O número de objetos deve ser o primeiro valor da linha **DATA** 200; será usado como índice nos comandos **DIM** referentes às matrizes de objetos — linha 180 — e para determinar o tamanho de diversos laços **FOR...NEXT**. O segundo índice da matriz **B$** é comprimento da maior descrição curta o segundo índice em **SI$**, da maior descrição longa.

Se usarmos linha **DATA** para cada objeto, programa ficará mais claro. No entanto, o número de objetos for muito grande, pode-se optar por colocar mais de um por linha. Em todo caso, ordem deve ser precisa, pois cada grupo de três dados contém elementos de três diferentes matrizes. A ordem é: número do local onde se encontra o objeto, descrição curta e descrição longa. Se o objeto só aparece mais tarde aventura — após ser encontrado, por exemplo, como olho — ou se surge ao acaso, como o fiscal da Receita, o número do local deve ser 0.

ROTINA DE AUXÍLIO

Esta é a última rotina a alterada. Verifique onde pode ser interessante "AJUDAR" o jogador. Use **IF...THEN** para fazê-lo.

Outras características do jogo original, tais como a rotina que faz surgir de repente o fiscal da Receita — linha 320 —, devem ser alteradas ou apagadas. Podemos também mudar o local onde começa aventura, na linha 280.

LISTA DE VARIÁVEIS

Para estudar em maior profundidade o nosso programa, use esta lista de variáveis.

G()	matriz contendo os números das linhas de descrição dos locais e de execução de ordens do jogador — execução dos verbos.
R$()	matriz com verbos e respostas do jogador.
R()	matriz com os números dos verbos.

Elementos correspondentes das duas matrizes acima referem-se mesmo verbo.

B()	matriz contendo o local onde cada objeto encontra.
B$()	matriz com a descrição curta dos objetos.
SI$()	matriz com as descrições longas.

Elementos correspondentes das três matrizes acima referem-se ao mesmo objeto.

NB	número de objetos da aventura.
L	posição do jogador.
LA	sinalizador de estado da lâmpada.
N,S,E,W	saídas de um local.
I$	resposta completa do jogador.
V$	verbo em **I$**.
N$	substantivo em **I$**.
I	contém um número correspondente a um grupo de verbos.
IN	número de objetos carregados pelo jogador.
A$	contém resposta à pergunta "Quer jogar novamente?"
G	contém o número do objeto deixado de lado pelo jogador — elemento **G** da matriz **B**.

EDIÇÃO NO TRS-80 E NO TRS-COLOR

- COMO TIRAR O MÁXIMO DO COMANDO EDIT
- MODIFICAÇÃO DE CARACTERES
- PROLONGAMENTO DE UMA LINHA
- PROCURA DE UM CARACTERE

■ todos os micros possuem comandos que facilitam a edição ■ programas ■ BASIC, ■ ■ comando EDIT das linhas TRS-80 ■ TRS-Color ■ um ■ mais sofisticados. Aprenda a utilizá-lo.

Como você já deve ter observado, é muito ■ um programa rodar corretamente pela primeira vez. Mensagens de erro, ou problemas que ocorrem durante a execução, tornam imperativo o teste repetido de um programa, até que todos os erros tenham sido eliminados.

Esse processo, chamado de *depuração*, é ■ geral trabalhoso e demorado, sobretudo se as linhas erradas tiverem que ser inteiramente digitadas. Quase todos os microcomputadores possuem algum tipo de editor de linhas para agilizar o processo de correção de programas. O sistema disponível nos micros das linhas TRS-80 e TRS-Color tem um funcionamento ■ pouco complicado, ■ ■ um dos mais completos. Neste artigo, você verá como explorar eficientemente ■ recursos de edição que seu micro lhe oferece.

Suponhamos, por exemplo, que você acabou de descobrir um erro de sintaxe na linha 20 de ■ programa. Para acionar o editor de linhas, digite **EDIT 20** ■ pressione ■ tecla **<ENTER>**. Nos computadores da linha TRS-Color, aparecerá logo abaixo a linha indicada e, em seguida, outra linha, contendo apenas ■ número ■ o cursor de posicionamento. Nos micros da linha TRS-80, o comando **EDIT** não mostrará a linha errada, ■ não ser que você pressione a tecla L uma vez.

```
20 X=RND(1
20 _
```

Como você pode notar, o erro consiste na falta do parêntese ■ direita.

O computador agora está em *modo de edição*, e você poderá fazer as alterações que quiser ■ linha apresentada na tela. Caso desista de editá-la, simplesmente pressione ■ tecla **<ENTER>** uma ■ ou, então, ■ tecla **E** (abreviatura de **END** — fim, em inglês), e o computador voltará ao *modo de comando.*

Uma vez que a linha desejada esteja pronta para ser corrigida, você poderá utilizar uma série de comandos de edição. É neste ponto que os principiantes correm o risco de ■ confundir, pois para ■ acionar ■ comandos pressiona-se uma única tecla alfabética (cujo resultado não aparece ■ tela) ou, então, um número de um ou mais dígitos, seguido da tecla alfabética.

No exemplo anterior, ■ correção consiste em acrescentar um parêntese ■ final da linha. Para isso, pressione ■ tecla **X**, que aciona o comando de e**X**tensão de linha; o cursor pulará automaticamente para o final da linha 20:

```
20 X=RND(1
20 X=RND(1_
```

Agora você pode digitar ■ parêntese que falta, pois ■ computador automaticamente entra em *modo* **I**, ■ de *inserção*. Para terminar a linha e sair do modo de edição, pressione **<ENTER>**. Não pressione **E**, pois o computador adicionará esta letra ■ final da linha: os comandos de edição só valem quando o modo **I** está desligado.

Vamos ver agora um caso mais complicado. Suponhamos que você queira modificar a linha seguinte, inserindo um novo caractere em algum ponto:

```
50 X=0:Y=0:W=363
```

O valor atribuído à variável **X** deveria ser, por exemplo, 20 e não 0, como está. Para fazer a modificação, digite **EDIT 50** ■ pressione **<ENTER>**. A linha 50 aparecerá na tela e o cursor logo em seu início. Você precisará deslocá-lo até ■ posição anterior ao primeiro zero, na linha, para poder inserir o novo caractere (no caso, um 2). Existem três maneiras de fazer ■ deslocamento. A primeira consiste em pressionar repetidamente a barra de espaço: cada vez que ela é pressionada, o cursor avança uma posição, e na tela aparece o caractere da posição seguinte. A segunda consiste em dizer ao editor quantos caracteres deve pular, digitando este número, seguido de uma pressão à barra de espaço. No terceiro método, pressiona-se a tecla S (de Search — procurar, em inglês), seguida do caractere para o qual você quer deslocar o cursor.

No exemplo acima, precisamos deslocar o cursor para a posição anterior

ao primeiro zero da linha. Bata duas vezes na barra de espaço ou, então, digite o número **2**, seguido de uma pressão ■ barra; se preferir, pressione a tecla ■ e, em seguida, a tecla **O**. Seja qual for ■ método utilizado, o cursor ficará posicionado logo após o sinal de igual. Agora, para inserir o número **2**, você deve entrar em modo de inserção. Para isso, pressione a tecla **I** uma vez; em seguida, acrescente o que quiser à linha. Digite normalmente as adições (pode-se usar, inclusive, o retrocesso). Pressione ■ tecla **<ENTER>** para sair do modo de edição, uma vez que tenha completado a inserção.

Caso você precise fazer outras correções na linha (por exemplo, acrescentar o número 1 antes do segundo zero, de modo a transformar em 10 o valor de **X**), deverá continuar no modo de edição. Para sair do modo de inserção ■ voltar ao modo de edição, pressione simultaneamente as teclas **<SHIFT>** e ↑. Com isso, você poderá deslocar o cursor até o segundo zero da linha, usando um dos três métodos explicados acima, digitar **I** para inserir e, finalmente, digitar o número **1**.

Algumas vezes, é necessário substituir caracteres, e não acrescentar outros. Eis aqui um exemplo:

```
70 PRINT "EDIYORA NOVA CULTURA
   L"
```

Chame ■ comando **EDIT** e posicione o cursor sobre ■ letra errada (Y). Para substituí-la pelo T, pressione uma vez a letra **C** (de Change — mudar, em inglês) e, em seguida, a letra correta. Se você quiser mudar mais de um caractere, acione novamente a tecla **C** ou, então, ■ número de caracteres, seguido da tecla **C**. Caso precise substituir, por exemplo, três letras consecutivas, coloque o cursor sobre a primeira ■ digite **3C**.

Para apagar uma letra, ou uma série de letras, coloque o cursor sobre o caractere que pretende apagar e pressione ■ letra **D**. Nos computadores da linha TRS-Color, ■ caractere indicado desaparecerá da tela e os demais serão deslocados para ■ esquerda, fechando o "buraco". No TRS-80, o caractere apagado será exibido entre dois sinais de exclamação. Para apagar vários caracteres, digite o número de caracteres que quer eliminar, seguido da tecla **D**. Colocando o cursor no primeiro E e pressionando **7D**, por exemplo, você apagará a palavra EDIYORA.

Também é freqüente a necessidade de apagar toda a parte final de uma linha. Suponhamos que você queira apagar ■ comando PRINT da linha abaixo:

```
10 X=89:Y=76:PRINT "X ■ Y DEFIN
   IDOS"
```

Entre ■ modo de edição ■ coloque o cursor até o ponto da linha a partir do qual se encontram ■ caracteres que deseja apagar (no exemplo, o segundo sinal de dois pontos). Em seguida, pressione a tecla **H** (abreviatura de **Hack**— podar, em inglês) e o resto da linha desaparecerá. Este comando também entra no modo de inserção, permitindo que você acrescente algo à linha ou que abandone o modo de edição, pressionando (ENTER).

Você pode também apagar partes de uma linha utilizando o comando ■ (de **Kill** — matar, em inglês). Ele elimina caracteres sucessivamente, até que seja encontrada a primeira ocorrência do caractere especificado. Se você digitar, por exemplo, **KD**, eliminará todos os caracteres entre a posição do cursor e a primeira ocorrência da letra D, na linha. Caso digite **3KD**, apagará tudo até a terceira ocorrência de D.

Depois de fazer algumas modificações em uma linha, liste-a, ainda dentro do modo de edição, para ver como ficou. Para isso, digite a tecla **L**.

Se você cometeu algum erro e deseja cancelar todas as modificações feitas, digite ■ tecla **A** (**Again** — novamente, em inglês); o cursor retornará ao seu início. O comando **Q** (**Quit** — abandonar, em inglês) faz ■ mesma coisa, mas sai do modo de edição.

Para se exercitar um pouco, digite o programa abaixo:

```
10 CLS
20 PRINT
30 PRINT "NU.MERO"."CUBO"
40 ■■ A=1 TO 13 STEP ■
50 PRINT A.*A*A
60 NEXT
70 PRINT "ESTE E' O FIM"
```

Ele contém alguns erros; seu aspecto final deverá ser ■ seguinte:

```
10 CLS
20 PRINT "TESTE"
30 PRINT "NUMERO"."CUBO"
40 FOR A=1 TO 13
50 PRINT A.A*A*A
60 NEXT
70 PRINT "FIM"
```

Para alterar o programa, procure usar todos os comandos de **EDIT** que aprendeu. Eis aqui a notação abreviada das modificações que poderá tentar:

```
EDIT 20 X"TESTE" [ENTER]
EDIT 30 S.D[ENTER]
EDIT 40 4[ESPAÇO]H[ENTER]
EDIT 50 S,IA[ENTER]
EDIT 70 SEKOD[ENTER]
```

Sumário dos comandos de EDIT

L	Lista ■ linha
C caractere	Muda caractere
nC caracteres	Muda os próximos n caracteres
I	Insere caracteres
D	Apaga um caractere
nD	Apaga n caracteres
H	Apaga o restante da linha e entra em modo de inserção
X	Entra em modo de inserção ■ final da linha
■ caractere	Procura ■ primeira ocorrência do caractere
nS caractere	Procura ■ n-ésima ocorrência do caractere
K caractere	Apaga tudo até a primeira ocorrência do caractere
nK caractere	Apaga tudo até ■ n-ésima ocorrência do caractere
<ESPAÇO>	Avança o cursor uma posição
n<ESPAÇO>	Avança o cursor n posições
←	Recua o cursor de uma posição
n ←	Recua o cursor de ■ posições
<SHIFT> ↑	Sai do modo de inserção e volta ao modo de edição
<ENTER>	Sai do modo de edição e inserção
E	Sai do editor
A	Retorna ao início da edição
■	Retorna ao início da edição ■ sai do editor.

ASSEMBLER PARA O MSX

Os montadores são programas tradutores encarregados de converter ■ programas-fonte, escritos em Assembly em programas-objeto, codificados em linguagem ■ máquina.

Como já foi dito em artigos destinados a outros computadores, montar um programa em linguagem de máquina, calculando os códigos a mão, pode ser bem cansativo. Ainda que se saiba de cor todos os códigos mnemônicos ■ seus equivalentes hexadecimais e ■ esteja familiarizado com ■ modos de endereçamento, o trabalho de tradução ■ transferência do programa para o computador é sempre tedioso e possível de erros.

Já que computadores são muito bons nesse tipo de trabalho, por que não usá-los para fazer ■ tradução? De fato, só teríamos a ganhar se fizéssemos isso, pois ■ mesmo programa poderia ainda ser usado para colocar o programa em código ■ memória.

O programa apresentado neste artigo é bem parecido com o Assembler para o ZX Spectrum. Isto se deve ■ fato de os dois computadores usarem o microprocessador Z-80.

Constituído em BASIC, ■ programa é bem mais lento do que os que são feitos em código de máquina (estes últimos podem ser facilmente encontrados ■ lojas especializadas). Todavia, ele funciona muito bem, embora devamos esperar algum tempo para que programas longos sejam montados ■ memória.

O ASSEMBLER

```
5000 CLS:KEYOFF:LOCATE8,10:PRIN
T"Um instante, por favor":CLEAR
500,&HDFFF:DIM K$(110),K(110),M
(110):H$="0123456789ABCDEF":B$=
"":G$="0123456789abcdef"
5010 DIMT$(100),R(100),Z$(100),
Z(100)
5020 B(1)=1:FORI=2TO9:B(I)=B(I
1)+B(I-1):NEXT
5030 DIMR$(8,4):FORJ=1TO4:FORI=
1TO8:READR$(I,J):R$(I,J)=LEFT$(
R$(I,J)+"    ",4):NEXTI,J
5040 DATA 0,1,2,3,4,5,6,7,nz,z
nc,c,po,pe,p,m,0,8,16,24,32,40
48,56,hl,ix,iy,bc,de,hl,sp,"
5050 DIMS$(8,2),T(18),U$(18):FO
RJ=1TO2:FORI=1TO8/J:READS$(I,J)
:S$(I,J)=LEFT$(S$(I,J)+"    ",4
):NEXTI,J:FORJ=1TO18:READT(J),U
$(J):NEXTJ
5060 DATA b,c,d,e,h,l,(hl),a,bc
,de,hl,sp,235,de,8,af,227,(sp)
60742,0,60758,1,60766,2,233,(hl
```

O que acontecerá se houver um erro em meu programa-fonte?

Nosso Assembler é capaz de emitir mensagens de erro, pois a sua estrutura de programação permite reconhecer certas falhas no programa em Assembly. Alguns comandos, por exemplo, só trabalham com ■■ registros a ■ hl. Se tentarmos usá-los com outros registros, o Assembler dirá: "Primeiro operando deve ser a ou hl".

Se um comando não for reconhecido devido ■ um erro de digitação, seremos informados: "Linha não reconhecida". A expressão "Deve haver dois operandos" significa que deixamos de digitar um número vital após ■ comando. "Operando incompatível" indica um uso inadequado do comando. "Primeiro operando deve ■■ sinalizador ou bit" significa que um operando inadequado foi empregado ■■ um comando de desvio ou de atribuição de bits.

```
),56809,(ix),65001,(iy),10,(bc)
,26,(de),60767,r,2,(bc),18,(de)
,60751,r,249,sp,60743,i,60759,i
5070 DEFFNB(X,I)=INT(X/B(I+1))-
INT(X/B(I+2))*2
5080 DEFFNX(X,I)=X-B(I+2)*FNB(X
,I)+B(I+1)
5090 DEFFNJ(X,I)=INT(X)-B(I+1)*
INT(X/B(I+1))
5100 DEFFNE(I$,J$)=(I$=LEFT$(J$
,LEN(I$)))
5110 FORI=1TO110:READK$(I),K(I)
,M(I):IFNOTFNE("*",K$(I))THENNE
XTI
5120 DATA ld,10,10,ld,26,10,ld,
60767,10,ld,60759,10,ld,2,138,l
d,18,138,ld,60751,138,ld,60743,
138,ld,64,6,ld,50,202,ld,58,74,
ld,249,139,ld,34,195,ld,42,67,l
d,60779,197,ld,60771,199,ld,97,
165,ld,64,54
5130 DATA adc,136,50,adc,60746,
3,add,128,50,add,9,149,and,160,
48,or,176,48,xor,168,48,nop,0,0
,sub,144,48,sbc,152,50,sbc,6073
8,3,cp,184,48,jp,130,45,jp,233,
9,jp,56809,9,jp,65001,9,jp,131,
49,jr,96,45,jr,88,41
5140 DATA call,132,45,call,141,
41,ret,201,0,djnz,74,40,dec,11,
17,dec,5,16,inc,3,17,inc,4,16,p
ush,197,17,pop,193,17,di,243,0,
ei,251,0,halt,118,0,ex,235,139,
ex,8,15,ex,227,143,exx,217,0
5150 DATA rát,199,132,rts,192,5
,bit,52032,20,defb,-256,40,ccf,
63,0,scf,55,0,cpl,47,0,cpd,6084
1,0,cpdr,60857,0,cpi,60833,0,cp
ir,60849,0,daa,39,0,im,60742,8,
im,60758,8,im,60766,8
5160 DATA in,60736,130,in,149,4
2,ind,60848,0,indr,60810,0,ini,
60840,0,inir,60850,0,ldd,60840,
0,lddr,60856,0,ldi,60832,0,ldir
,60848,0,neg,60740,0,otdr,60859
,0,otir,60851,0,out,60737,2,out
,141,170,outd,60843,0,outi,6083
5,0
5170 DATA res,52096,20,reti,607
49,0,retn,60741,0,rl,51984,64,r
la,23,0,rlc,51968,16,rlca,7,0,r
ld,60783,0,rr,51992,64,rra,31,0
,rrc,51976,16,rrca,15,0,rrd,607
75,0
5180 DATA set,52160,20,sla,5200
0,16,sra,52008,16,srl,52024,16,
defw,-256,41
5190 DATA "*",0,0:II=I:K(110)=I
■
5200 CLS:PRINTTAB(15)"ASSEMBLER
":LOCATE10,5:PRINTTAB(10)"(c)
Ler na fita cassete":PRINT:PRIN
TTAB(10)"(g)  Gravar na fita":P
RINT:PRINTTAB(10)"(e)  Edição"
5210 PRINT:PRINTTAB(10)"(m)  Mo
ntar":PRINT:PRINTTAB(10)"(a)  A
pagar linha":PRINT:PRINTTAB(10)
"(l)  Listar":PRINT:PRINTTAB(10
)"(s)  Saída"
5220 A$=INKEY$:IFA$=""THEN5220
5230 JJ=INSTR("cgemals",A$)
5240 IFJJ=0THENPRINT"<"A$"> ???
?":FORJ=1TO500:NEXT:GOTO5200
5250 CLS:ONJJGOSUB10420,10450,1
0490,5290,10710,10760,10900
5260 PRINT:PRINT"Qualquer tecla
 para continuar"
5270 A$=INKEY$:IFA$=""THEN5270
5280 GOTO5200
5290 FORG=1TO100:R(G)=G-1:NEXTG
:FH=100
5300 K0=0:K9=99:P0=0:VV=0
5310 K=K0:P=P0
5320 GOSUB8000
5330 GOSUB7000:O$=I$:IFLEFT$(O$
,1)="*" THENPRINTO$;:GOTO5320
5340 IFO$="end"THENPRINT:PRINT"
    FIM. Endereço final = ";P-1
5350 IFO$="end"THENP0=P:RETURN
5370 IFO$<>"org"THEN5400
5380 GOSUB7000:S=0:IFLEFT$(I$,1
)="*"THENS=P:I$=RIGHT$(I$,LEN(I
$)-1)
5390 P=VAL(I$)+S:PRINT"    org
";P;:GOTO5320
5400 IFP=0THENPRINT" (falta org
)":P=&HE000
5410 P$=O$+"!":FORI=1+18*ABS(O$
<>"ld")TO110:IFO$<=K$(I)ANDP$>K
$(I)THEN5500
5420 NEXTI:PRINTO$
5430 IFLEFT$(I$,1)="."THENI$=RI
GHT$(I$,LEN(I$)-1)
5440 GOSUB9000:GG=R(G)
5450 IFABS(GG)<=100THENS=SGN(Z(
GG)):B=INT(ABS(Z(GG))/65536!):R
=ABS(Z(GG))-B*65536!:Q=PEEK(R)+
256*PEEK(R+1):POKER,FNJ(P*S+Q,8
):PRINT" colocando ";FN J(P*S+Q
,8);" ■■ ";R:IFBTHENPOKE(R+1),F
NJ((P*S+Q)/256,8):PRINT" coloca
ndo ";FNJ((P*S+Q)/256,8);" ■■ "
;R+1
5460 IFABS(GG)<=100THENGH=R(GG)
:R(GG)=FH:FH=GG:GG=GH:GOTO5450
5470 IFI$=""THENR(G)=P+100:GOTO
5330
5480 PRINT" (Linha não reconhec
ida)"
5490 GOTO5420
5500 Z=0:R=0:E=0:PRINT"    ";O$
;
5510 OP=K(I):IFM(I)=0THEN6090
5520 GOSUB7000:A$=I$:PRINT" ";A
$;
5530 M=M(I):OP=K(I):B=FNB(M,0):
B7=B+2*FNB(M,7)+1:Z=0:IFFNJ(M,3
)<2THENC$=A$:GOTO5720
5540 FORJ=1TOLEN(A$):IFMID$(A$,
J,1)=","THEN5580
5550 NEXTJ:IFO$="ret"ORO$="rts"
THEN5580
5560 IFFNE(O$,K$(I+1))THENI=I+1
:GOTO5530
5570 PRINT" (são necessários do
is operandos)":GOTO5320
5580 B$=LEFT$(A$,J-1):C$=RIGHT$
(A$,LEN(A$)-J)
5590 IFFNB(M,2)THEN5650
5600 IFFNB(M,7)THEND$=C$:C$=B$:
B$=D$
5610 IFB$=MID$("ahl",B+1,B+1)TH
EN5720
5620 IFB$="(c)"AND(O$="in"ORO$=
"out")THEN5720
5630 IF(FNE(O$,K$(I+1)))AND(FNJ
(M(I+1),3)>=2)THENI=I+1:GOTO553
0
5640 PRINT" (primeiro operando
deve ser a ou hl)":GOTO5320
5650 IFFNB(M,1)THEN5690
5660 E$=LEFT$(B$+"    ",4):FORJ=
1TO8:IFE$=R$(J,B7)THENOP=OP+8*(
J-1)*ABS(B7<4)+16*(J-6)*ABS(B7=
4)*ABS(J>3):Z=(J-1)*ABS(B7=4)*A
BS(J<=3):GOTO5710
5670 NEXTJ:IFP$>K$(I+1)AND(FN J
(M(I+1),3)>=2)THENI=I+1:GOTO553
■
5680 PRINT" (primeiro operando
deve ser sinalizador ou bit)":G
OTO5320
5690 IFFNB(M,7)THEND$=C$:C$=B$:
B$=D$:GOTO5660
5700 X=8:GOSUB5750:IFETHEN5730
5710 IFC$=""THEN6090
5720 X=1+15*B+7*ABS(OP<=6ANDOP>
=40RB$="(c)"):B$=C$:GOSUB 5750:
IFABS((E=0)*NOTE)THEN6090
5730 IFE=2ORP$>K$(I+1)ANDFNJ(M(
I+1),3)=FNJ(FNX(M,0),3)THENE=0:
I=I+1:GOTO5530
5740 GOTO5320
5750 R=0:IFFNB(M,4)ANDFNE(LEFT$
("(",ABS((B=0)*NOTB)),B$)THENZ2
=ABS(FNE("ix",RIGHT$(B$,LEN(B$)
+B-1)+" "))+2*ABS(FNE("iy",RIGH
T$(B$,LEN(B$)+B-1)+" ")):IFZ2TH
ENZ=Z2:E$=LEFT$(B$,LEN(B$)-ABS(
(B=0)*NOTB)):B$=MID$("(hl)",1+B
,4-2*B):F$="0"+RIGHT$(E$,LEN(E$
)+B-3)
5760 IFFNB(M,3)THEN5790
```

```
5770 ES=LEFT$(B$+"    ",4):FORJ=
1TO8/(B+1):IFES=S$(J,B+1)THENOP
=OP+(J-1)*X:RETURN
5780 GOTO5810
5790 J2=9+9*ABS(OS="ld"):FORJ=J
2-8TOJ2:IFK(I)<>T(J)THEN5810
5800 IFFNE(B$,U$(J))THENRETURN
5810 NEXTJ:IFB$="af"THENIFFNE("
p",O$)THENOP=OP+48:RETURN
5820 IFFNB(M,6)ANDFNE("(",B$)TH
ENB$=MID$(B$,2,LEN(B$)-2):GOTO5
860
5830 IFFNB(M,5)THENOP=FNX(OP+6*
ABS((B=0)*NOT B),6):GOTO5860
5840 IFP$>K$(I+1)THENE=2:RETURN
5850 PRINT" (operando incompatí
vel)":E=1:RETURN
5860 R=65536!
5870 S=1
5880 IFB$=""THEN6080
5890 X$=LEFT$(B$,1):D$=RIGHT$(B
$,LEN(B$)-1):IFX$="*"THENR=R+P*
S:B$=D$:GOTO5870
5900 IFX$="+"THENB$=D$:GOTO5880
5910 IFX$="-"THENB$=D$:S=-S:GOT
O5880
5920 IFX$="'"THENR=R+ASC(D$)*S:
B$=RIGHT$(D$,LEN(D$)-1):GOTO587
0
5930 Q=0:IFX$<>"%"ORD$<"0"ORD$>
="2"THEN5960
5940 IFD$>="0"ANDD$<"2"THENQ=Q*
2+ASC(D$)-48:D$=RIGHT$(D$,LEN(D
$)-1):GOTO5940
5950 R=R+Q*S:B$=D$:GOTO5870
5960 IFX$<>"$"ORD$<"0"ORD$>="g"
THEN6000
5970 X$=CHR$(ASC(D$)):FORG=0TO1
5:IFX$<>MID$(H$,G+1,1)ANDX$<>MI
D$(G$,G+1,1)THEN5990
5980 Q=Q*16+G:D$=RIGHT$(D$,LEN(
D$)-1):GOTO5970
5990 NEXTG:R=R+Q*S:B$=D$:GOTO58
70
6000 IFX$<"a"ORX$>"z"THEN6040
6010 I$=B$:GOSUB9000:IFI$<>""TH
EN6010:GOSUB9400
6020 IFR(G)<>23000ANDABS(R(G))>
100THENR=R+(R(G)-100)*S:B$=I$:G
OTO5870
6030 IFR(G)=23000ORABS(R(G))<=1
00THENGH=R(FH):R(FH)=R(G):R(G)=
FH:FH=GH:Z(R(G))=(P+SGN(OP)+ABS
(ABS(OP)>255)+2*ABS(Z>0)+65536!
*ABS(((ABS(BORFNB(M,6))<>0)*1)A
ND(O$<>"jr")))*S:B$=I$:GOTO5870
6040 IFX$<"0"ORX$>"9"THENR=0:GO
TO6070
6050 IFB$>="0"ANDB$<":"THENQ=Q*
10+ASC(B$)-48:B$=RIGHT$(B$,LEN(
B$)-1):GOTO6050
6060 R=R+S*Q:GOTO5870
6070 PRINT" (endereçamento invá
lido)"
6080 R=R-(P+2)*ABS(O$="djnz"ORO
$="jr"):RETURN
6090 PRINTTAB(16);:BY=P/256:GOS
UB6190: BY=P:GOSUB6190:GOSUB616
0
6100 IFZTHENBY=189+Z*32:GOSUB61
80:GOSUB6160
6110 IFOP>=0THENBY=OP/256:GOSUB
6170:GOSUB6150:BY=OP:GOSUB6180
6120 IFR=0THEN5320
6130 GOSUB6160:BY=R:GOSUB6180:I
F(ABS(BORFNB(M,6))<>0)ANDO$<>"j
r"THENBY=R/256:GOSUB 6180
6140 GOTO5320
6150 IFZ<>0ANDBY<>0ANDABS((B=0)
*NOTB)<>0THENGOSUB6260:BY=VAL(F
$):GOSUB6180:Z=0
6160 PRINT" ";:RETURN
6170 IFINT(BY)<=0THENRETURN
6180 BY=FNJ(BY,8):POKEP,BY:P=P+
1
6190 BY=FNJ(BY,8):PRINTMID$(H$,
1+INT(BY/16),1);MID$(H$,FNJ(BY,
4)+1,1);
6200 RETURN
7000 IFK>NTHENI$="end":RETURN
7010 K1=K9+1:IFK9>=LEN(T$(K))TH
ENI$="/faltando/":RETURN
7020 K9=K1:IFMID$(T$(K),K1,1)="
 "THEN7010
7030 IFK9>LEN(T$(K))THENI$=RIGH
T$(T$(K),LEN(T$(K))-K1+1):RETUR
N
7040 IFMID$(T$(K),K9,1)<>" "THE
NK9=K9+1:GOTO7030
7050 I$=MID$(T$(K),K1,K9-K1):RE
TURN
8000 IFK>0THENIFRIGHT$(T$(K),LE
N(T$(K))-K9+1)>T$(99)THENPRINTR
IGHT$(T$(K),LEN(T$(K))-K9+1);
8010 K=K+1:K9=0
8020 PRINT:RETURN
9000 X$=""
9010 IFI$<"a"ORI$>"z"THEN9030
9020 X$=X$+LEFT$(I$,1):I$=RIGHT
$(I$,LEN(I$)-1):GOTO9010
9030 IFI$<>""THENRETURN
9400 FORG=1TOVV:IFFNE(X$,Z$(G))
THENRETURN
9410 NEXTG:VV=VV+1:Z$(VV)=X$:G=
VV:R(G)=23000
9420 RETURN
10420 CLS:LOCATE0,10:PRINT"Posi
cione a fita, pressione qualque
r   tecla e aperte PLAY no grav
ador"
10430 U$=INKEY$:IFU$=""THEN1043
0
10440 OPEN"CAS:ASM"FORINPUTAS#1
:CLS:LOCATE6,10:PRINT"Carregand
o programa fonte":INPUT#1,N:FOR
J=1TON:LINEINPUT#1,T$(J):NEXT:C
LOSE#1:RETURN
10450 CLS:LOCATE0,10:PRINT"Posi
cione a fita, aperte RECORD no
    gravador e pressione qualqu
er tecla"
10460 U$=INKEY$:IFU$=""THEN1046
0
10470 OPEN"CAS:ASM"FOROUTPUTAS#
1:CLS:LOCATE7,10:PRINT"Gravando
 programa fonte":PRINT#1,N:FORJ
=1TON:PRINT#1,T$(J):NEXT:CLOSE#
1:RETURN
10490 PRINT"Qual o número da li
nha (múltiplo de 10)";
10500 INPUTK:CLS
10510 K2=K/10:IFK2>NTHENK2=N+1:
N=N+1:T$(K2)=""
10520 IFK2<.1THENK2=.1
10530 IFK2=INT(K2)THEN10550
10540 K2=INT(K2)+1:FORK3=NTOK2-
1STEP-1:T$(K3+1)=T$(K3):NEXT:N=
N+1:T$(K2)=""
10550 P1=887:P0=P1
10560 LOCATE0,21:PRINTK;TAB(6)T
$(K2)+STRING$(10,32);:P9=P0+LEN
(T$(K2))
10570 IFP1<P0THENP1=P0
10580 IFP1>P9THENP1=P1-1
10590 VPOKEBASE(0)+P1-1,32:VPOK
EBASE(0)+P1,195
10600 P7=0:A$=INKEY$:IFA$=""THE
N10600
10610 IFA$=CHR$(13)THEN10700
10615 IFA$=CHR$(27)THENRETURN
10620 IFA$=CHR$(28)THENVPOKEBAS
E(0)+P1,32:P1=P1+1:GOTO10580
10630 IFA$=CHR$(29)THENVPOKEBAS
E(0)+P1,32:P1=P1-1:GOTO10570
10640 IFA$=CHR$(30)THENA$="":GO
TO10670
10650 IFA$=CHR$(31)THENA$=" "+M
ID$(T$(K2),P1-P0+1,1):P7=-1:GOT
O10670
10660 IFA$<" "THEN10600
10670 IFP1-P0+1>LEN(T$(K2))THEN
T$(K2)=LEFT$(T$(K2),P1-P0)+A$:G
OTO10690
10680 T$(K2)=LEFT$(T$(K2),P1-P0
)+A$+RIGHT$(T$(K2),LEN(T$(K2))-
P1+P0-1)
10690 P1=P1-(LEN(A$)>0)+P7:GOTO
10560
10700 LOCATE0,24:PRINT:K=K+10:G
OTO10510
10710 IFN=0THENCLS:PRINT" Nada
a apagar":RETURN
10720 CLS:PRINT"Qual o número d
a linha (múltiplo de 10)";
```

MICRO DICAS

COMO ENCONTRAR ERROS EM PROGRAMAS LONGOS

O erro mais comum em programas como o desta lição consiste em deixar de digitar um segmento de linha **DATA** (nisso também o MSX se assemelha ao ZX Spectrum). Assim, quando surge a mensagem "OUT OF DATA", devemos averiguar o que está faltando nas linhas **DATA**. Até mesmo a falta de uma vírgula causará problemas. Se o seu Assembler não funcionar na primeira vez, não se preocupe: o MSX possui uma função de rastreamento (*trace*) que o ajudará a encontrar os erros.

Para ativá-la, digite **TRON** no modo imediato, sem número de linha. Depois rode o programa usando **RUN**. O número da linha BASIC que estiver sendo executada nesse momento aparecerá na tela. Esse recurso torna mais fácil a detecção de erros nos programas mais longos.

A função é desativada pelo comando **TROFF**.

```
10730 INPUTK:K2=K/10
10740 IFK2>NORK2<1ORK2<>INT(K2)
THENPRINT"Esta linha não existe
":RETURN
10750 K=K2:FOR K3=K2 TO N:T$(K3
)=T$(K3+1):NEXT:N=N-1:LOCATE 0,
10:PRINT K*10;"  ";T$(K):RETURN
10760 IFN=0THENPRINT" Nada a li
star":RETURN
10770 PRINT"Quais ■■ números da
 primeira ■ da      última linh
■ (múltiplos de 10)";
10780 INPUTK,K2:K=INT(K):K2=INT
(K2):K1=K/10:K2=K2/10
10790 IFK2>NTHENK2=N
10800 IFK1<1THENK1=1
10810 IFK2<K1ANDK2=NTHENRETURN
10820 IFK2<K1THENCLS:PRINT"Núme
ros de linha inválidos":GOTO107
70
10830 CLS:FORK3=K1TOK2:PRINTK3*
10;"  ";T■(K3):NEXT
10840 RETURN
10900 END
```

Os espaços em branco foram eliminados de modo ■ diminuir a quantidade de memória ocupada pelo programa. Algumas linhas — 5450 e 5750, por exemplo — são quase do tamanho máximo permitido pelo computador; assim, qualquer espaço adicional pode fazer com que a porção final dessas linhas seja ignorada.

As linhas 5030, 5050, 5390, 5500 apresentam, cada uma, quatro caracteres ■■ branco entre as aspas; ■■ linhas 5660 e 5770, três; e as linhas 10750 ■ 10830, dois.

COMO FUNCIONA O PROGRAMA

Antes de montar o programa devemos proteger uma área de memória, onde o Assembler colocará ■■ códigos por meio do comando **CLEAR**. Isso é feito na linha 5000, que reserva 500 bytes para os cordões (strings) e protege ■ memória a partir de E000 hexadecimal. Mude esse valor conforme ■■ necessidades. Programas grandes podem exigir maior espaço de cordões.

Depois de digitar, rode o Assembler e verá um menu. Para digitar um programa em Assembly, tecle "e" (não se esqueça de destravar ■ tecla **CAPS**, para que o MSX aceite letras minúsculas). O computador pedirá então o número da linha. Cada instrução deve ser introduzida em uma linha numerada, seme-

lhante ao que ocorre no BASIC. É preciso que os números das linhas sejam múltiplos de 10; apenas um mnemônico Assembly com seus operandos pode ser introduzido em cada linha.

A primeira linha de seu programa deve ser mais ou menos assim:

```
10  org -8192
```

– 8192 ■ ■ endereço inicial do programa em código. Obviamente, ele deve estar ■ área protegida da memória. Se esquecermos de definir **org**, ou seja, ■ origem, o Assembler colocará o programa a partir de DFFF, em hexadecimal (ou – 8192, em decimal).

O programa aceita os códigos mnemônicos padrão do chip Zilog Z-80, com exceção dos comandos de retorno condicional. O comando de retorno incondicional de sub-rotinas, cujo ■ mônico é **ret**, entretanto, *funciona* em nosso Assembler.

Algumas vezes, porém, podemos utilizar retornos condicionais, como **ret nz**, que significa "retorne se não for zero". Em nosso caso, use: **rts** ■ (**rts** deve ser usado no lugar de **ret** sempre que precisarmos empregar retornos condicionais). Quando ■ trata de retorno incondicional — ou seja, **ret** não seguido de nenhuma letra — use ■ mnemônico padrão **ret**.

No artigo da página 213 (Programas ■ *Código de Máquina*), usamos o mnemônico **outir**, quando na verdade o padrão Z-80 é **otir**. Nosso Assembler aceita **otir**.

Números hexadecimais devem ser precedidos de **$**. Números binários, de **%**. Qualquer número sem nada na frente será interpretado como decimal. Qualquer expressão que não seja ■ comando Assembly será considerada como um rótulo *(label)*. Evite usar palavras parecidas ou muito longas; números não são permitidos.

Para que ■ Assembler saiba onde parar, termine seu programa com **end**.

Antes de apertar o **ENTER** para editar uma linha pressione as teclas do cursor. As setas "direita" ■ "esquerda" movem o cursor ao longo da linha; a seta "para baixo" serve para inserir ■aracteres e a ■ "para cima", para apagá-los. A edição de linhas é bastante lenta ■ muitas vezes os caracteres digitados demoram a aparecer ■ vídeo.

Use a tecla **ESC** para sair do modo de edição. Essa tecla deve substituir ■ comando **ENTER**, depois que a última linha tiver sido digitada; ■ contrário, uma linha em branco será incorporada ■ programa ■ Assembly.

De volta ao menu, pressione "l" para obter uma listagem do programa em Assembly ■ tela.

Se houver algum erro ■ introdução do programa-fonte, volte ■ menu ■ pressione "e"; forneça então o número da linha que deseja mudar.

Por outro lado, se quisermos inserir uma linha entre duas já existentes, bastará digitá-la usando um número intermediário, quando no modo de edição. Assim, ■ listarmos o programa novamente, veremos que ele foi renumerado com ■ nova linha no lugar certo. Todavia, apenas uma linha pode ser introduzida dessa maneira de cada vez.

Para apagar uma linha, é preciso voltar ■ menu, teclar "a", e fornecer o número da linha que queremos eliminar.

Quando estivermos satisfeitos com ■ programa Assembly a ser montado (*programa-fonte*), devemos voltar ao menu e teclar "m" para que ele seja "montado" na memória. Ao mesmo tempo, obteremos ■ listagem dos códigos equivalentes na tela (*programa-objeto*).

Se ■ esta altura notarmos um erro em alguma das linhas, devemos voltar ■ menu e editar a linha correspondente.

Uma vez montado o programa, ■ endereço final da rotina em código aparecerá ■ tela.

A opção de gravação — "g", no menu — armazena ■ programa-fonte em fita cassete — os mnemônicos, portanto. Para gravar ■ Assembler, use a via normal — **CSAVE**.

Para executar o programa em código, utilize instruções do tipo DEF **USRO** ■ **A = USRO**. Grave ■ programa-objeto, empregando:

```
BSAVE "CAS:nome", endereço
inicial, nº de bytes
```

"Nome" é a denominação do programa-objeto ■ deve estar entre aspas. **BSAVE** informa ao computador que está sendo gravado um programa em código ■ não em BASIC. O endereço inicial é a origem do seu programa na memória. Podemos calcular o número de bytes subtraindo a origem do endereço final e somando 1. Podemos ainda acrescentar ■ endereço inicial para execução (não mostrado ■ exemplo). Como esse endereço é quase sempre igual ao endereço inicial, ■ inclusão é opcional.

UM TESTE

Para testar seu programa, entre o programa de deslocamento horizontal da tela para ■ direita que se encontra na página 215. Uma vez traduzido para código, ele ficará assim:

```
11 C0 ■ 21 ■ ■ ■ 18 C5 1A 01
 27 00 ED B8 12 2B 1B C1 10 F3
C9
```

Note que devemos usar letras minúsculas para digitar o programa-fonte. Nosso Assembler não reconhece comandos em maiúsculas.

GRÁFICOS INSTANTÂNEOS

Você não precisa conhecer código de máquina nem entender de sistema binário para animar ■ jogos com figuras simples. Leia este artigo e veja como ■ fácil fazer isso.

Qualquer pessoa com algumas horas de experiência com computadores pode criar caracteres gráficos originais para usar em jogos. Tudo o que se precisa é lápis e papel para esquematizar as idéias, além de duas (ou, no máximo, três) rotinas simples para transformar tais idéias em figuras no computador.

Cada tipo de máquina exige um método particular para ■ criação de novos padrões de caracteres gráficos definidos pelo usuário. O tamanho desses caracteres também é variável. O MSX tem sprites de 16 por 16 pixels (ou pontos), enquanto o Spectrum oferece apenas um UDG de 8 por ■ pixels (os compatíveis com o ZX-81 não serão tratados aqui).

No entanto, independentemente do que seu computador ofereça, o mais conveniente é começar criando figuras de 8 ■ ■ (tamanho aproximado de um "inimigo" ■ um jogo espacial). Uma vez que você tenha aprendido a trabalhar com ■ padrão, será fácil criar figuras maiores, ou mesmo encadear duas ou mais pequenas figuras para formar uma terceira.

DO DESENHO PARA O BINÁRIO

Em alguns computadores, os dados necessários para definir um UDG podem ser fornecidos diretamente na forma binária. Em outros, você terá que convertê-los em decimal ou em hexadecimal. Assim, leia primeiramente a seção para sua máquina antes de fazer qualquer conversão. No Apple você terá que usar o nosso editor de figuras.

Os computadores armazenam as informações que lhes são dadas em binário. Mas você não precisa conhecer esse sistema numérico para transformar seus desenhos em filas de números binários. Tudo o que deve fazer para isso é:

— Utilizar o número 1 sempre que quiser obter um ponto.
— Empregar o número 0 toda vez que quiser um espaço.

Tomemos como exemplo o padrão da cruz de Lorena. A linha do topo é composta de três espaços, um ponto ■ mais quatro espaços. Em binário, 00010000.

A segunda linha é formada por dois espaços, três pontos e três espaços: 00111000. Todo o padrão pode ser representado assim:

```
0 0 0 1 0 0 0 0
0 0 1 1 1 0 0 0
0 0 0 1 0 0 0 0
0 1 1 1 1 1 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
```

DE BINÁRIO PARA DECIMAL

A maneira mais rápida de converter binário em decimal é usar a pequena tabela de nove linhas por oito colunas que apresentamos ■ seguir. Na primeira linha, escreva os números 128, 64, 32, 16, 8, 4, 2, 1. Nas outras oito coloque ■ números correspondentes ao gráfico que você quer. Aqui temos, por exemplo, a tabela para a cruz de Lorena:

128	64	32	16	8	4	2	1
0	0	0	1	0	0	0	0
0	0	1	1	1	0	0	0
0	0	0	1	0	0	0	0
0	1	1	1	1	1	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0

Para fazer a conversão, ignore os zeros. Inicialmente, compare os 1 em binário com o número no topo da coluna. Some então todos os números correspondentes aos 1 da primeira linha. Repita essa operação em todas as linhas.

No exemplo, ■ primeira linha contém o número 1 apenas na quarta coluna. Portanto, temos 16.

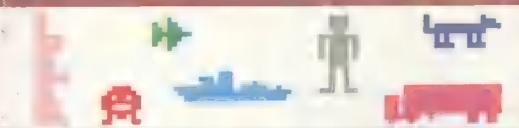
A segunda linha tem três 1 que correspondem ■ 32, 16 ■ 8. Soma: 56.

Quando o processo de verificação houver terminado, você terá oito números decimais. A declaração **DATA** necessária para entrar ■ dados no computador ficará assim:

```
DATA 16,56,16,124,16,16,16,0
```

De início, você pode ficar com ■ impressão de que esta é uma forma muito complicada de fazer o trabalho. No entanto, depois de meia dúzia de tentativas, seu ritmo de trabalho se tornará suficientemente rápido. Além disso, as combinações mais comuns, como 255 (ou 11111111 em binário), serão logo memorizadas, poupando-as da necessidade de fazer cálculos para chegar ■ sistema decimal.

DE BINÁRIO PARA HEXADECIMAL

Converter números binários em he-

■ TRANSFORME DESENHOS EM FILAS DE NÚMEROS BINÁRIOS
■ CONVERSÕES DE BINÁRIO PARA DECIMAL E PARA HEXADECIMAL
■ USE GRÁFICOS EM PROGRAMAS

xadecimais é ainda mais fácil. Para isso, basta aplicar a tabela ■ seguir, não sendo necessário nem mesmo escrever nela os números binários:

Binário	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Desta vez não ignore os zeros. Para começar, tome os primeiros quatro dígitos do número binário e procure o seu equivalente ■ hexa. Em seguida, faça o mesmo com os quatro dígitos seguintes. A reunião dos dois dígitos hexadecimais obtidos define o número que você precisa.

Voltando ■ cruz de Lorena, ■ primeira metade do binário da primeira linha ■ 0001. Em hexa temos, pela tabela, 1. A segunda metade é 0000. Em hexa, temos 0. Escreva os dois números juntos e teremos 10, ■ hexa correspondente ao binário 00010000.

De forma similar, na segunda linha temos 0011 (que é 3 em hexa) e 1000 (8 em hexadecimal). Assim, ■ número hexadecimal completo é 38.

Repita ■ processo até a última linha ■ terá oito números hexadecimais. A sua declaração **DATA** completa ficará como segue:

```
DATA 10,38,10,7C,10,10,10,00
```

Além disso, a única coisa a fazer é dizer ao computador (que não pode adivinhar) se o número que você está digitando é decimal ou hexa. Veja como fazer isto na seção específica do seu computador.

Evidentemente, é possível escrever um pequeno programa para fazer ■ conversão de bases numéricas, mas ele não será muito útil se você quiser modificar alguma informação que já esteja na memória do computador.

T

O TRS-Color aceita **DATA** em decimal, hexa ou binário.

Se os dados (**DATA**) estão em hexa, você terá que adicionar uma linha extra ao programa para convertê-los em decimal (veja ■ seguir).

O primeiro programa desenha um pequeno avião no canto superior esquerdo do vídeo. Este pode não ser ■ melhor lugar para vê-lo; mas é o mais fácil para começar, se você quiser fazer um programa para movimentá-lo pela tela.

```
20 PMODE 4,1
30 PCLS
40 SCREEN 1,1
60 FOR L=0 TO 7
70 READ N$
■■ POKE L*32+1536,VAL("&H"+N$)
90 NEXT L
110 GOTO 110
500 DATA 00,10,18,9C,FF,9C,18,1
■
```

Digite o programa e execute-o.

PMODE 4,1 foi escolhido na linha 20 porque só se pode produzir gráficos UDG quando neste modo de resolução gráfica mais alta.

Para limpar a tela sobre ■ qual vai ■ desenhar, deve-se usar o comando **PCLS** como na linha 30. Ele se aplica não só ao **PMODE 4,1**, como a todos os modos de alta resolução.

Use **SCREEN 1,1**, como na linha 40, para ligar a tela de alta resolução de forma a mostrar o seu UDG. **SCREEN 1,1** também seleciona as cores branca e preta.

O laço **FOR...NEXT** das linhas 60 ■ 90 faz com que ■ linha 70 seja executada oito vezes. Toda vez que encontra **READ N$**, o computador lê o dado seguinte da linha **DATA** de número 500.

A linha 80 é importante por duas razões. A primeira é que ela faz com que o padrão de pontos definido pela linha 500 apareça no vídeo. A segunda ■ que ela converte os números hexadecimais da linha **DATA** em decimais ■ coloca-os diretamente na área de memória que controla o que aparece na tela do computador.

Finalmente, a linha 110 é um laço que mantém a alta resolução. Sem essa linha o programa terminaria, provocando o retorno automático para ■ modo texto; em conseqüência, você não veria o seu desenho na tela.

GRÁFICOS UDG MAIS ALTOS

Você pode criar um UDG "alto e magro", em vez de um 8x8, mudando a linha 60 ■ alterando os dados da linha 500. Em primeiro lugar, modifique a linha 60 para:

```
60 FOR L=0 TO 7
```

Em seguida, mude ■ linha 500 de forma que fique assim:

```
500 DATA 00,10,18,9C,FF,9C,18,1
0
```

A alteração na linha 60 permite ■ leitura de um maior número de dados da linha 500. E esta define a imagem de um coelho, usando hexadecimais. Assim, ■■ executar o programa, você encontrará a figura do coelho na tela.

Esse método permite ■ criação de gráficos de qualquer tamanho. Se você já desenhou a ■■■ figura em papel quadriculado, conte quantas linhas foram usadas. Altere ■ linha 60 usando o número de linhas do desenho. A seguir, faça com que o número de informações da linha **DATA** corresponda ao número de vezes que ■ laço **FOR...NEXT** for executado.

GRÁFICOS MAIS LARGOS

O desenho de figuras baixas e largas (em vez de altas e estreitas) exige um pouco mais de trabalho.

Comece mudando as linhas 60 e 80 da seguinte forma:

```
60 FOR L=0 TO 7
80 POKE L*32+1536+F,VAL("&H"+N$
```

Altere a linha **DATA** de forma que apareça assim:

```
500 DATA 0F,0F,EF,EF,EF,EF,FE,4
4,FF,FF,FF,FF,FF,FF,40,00,FF,FF
,FF,FF,FF,FF,66,66
```

Finalmente, adicione estas linhas:

```
50 FOR F=0 TO 2
100 NEXT F
```

Quando você executar o programa, verá na tela a figura de um caminhão de carga.

Como isto funciona? Os 24 trechos de informação da linha 500 formam três quadrados de oito pixels de lado. Se os dados fossem lidos por um único **FOR...NEXT** como antes, o caminhão apareceria de forma peculiar, cortado em três pedaços que se amontariam no vídeo. Assim, o computador deve ser avisado para colocar os blocos lado a lado. Para fazer isso, um laço extra altera o valor do **POKE** na linha 80, de forma que o segundo bloco apareça na linha do topo, ao lado do primeiro. Depois de ser lido, o terceiro bloco é colocado ao lado dos outros dois por intermédio do comando **POKE**.

MOVIMENTE OS

Você pode mover a figura do avião pela tela usando estas linhas:

```
110 DIM A(3),B(3)
120 GET (0,0)-(7,7),A,G
130 PCLS
140 LET X=127
150 LET Y=95
160 PUT (X,Y)-(X+7,Y+7),A,PSET
170 LET LX=X
180 LET LY=Y
190 IF PEEK(338)=251 AND Y>2 TH
EN Y=Y-2:GOTO 240
200 IF PEEK(342)=253 AND Y<182
THEN Y=Y+2:GOTO 240
210 IF PEEK(340)=247 AND X>3 TH
EN X=X-3:GOTO 240
220 IF PEEK(338)=247 AND X<245
THEN X=X+3:GOTO 240
230 GOTO 190
240 PUT (LX,LY)-(LX+7,LY+7),B,P
SET
250 GOTO 160
```

Quando você executar este programa, a tecla Z movimentará a figura para a esquerda; X o fará para a direita; P para cima; e L para baixo.

O comando **GET** permite ao computador lembrar-se do que está em um determinado ponto da tela, enquanto **PUT** coloca o que foi encontrado em qualquer outro lugar. **DIM**, por sua vez, reserva espaço na memória para **GET**.

As linhas 190 até 220 verificam se uma tecla foi pressionada e movem o UDG em caso positivo.

Se você quiser mover o coelho deve fazer as seguintes alterações:

```
110 DIM A(6),B(6)
120 GET (0,0)-(7,23),A,G
160 PUT (X,Y)-(X+7,Y+23),A,PSET
200 IF PEEK(342)=253 AND Y<166
THEN Y=Y+2:GOTO 240
240 PUT (LX,LY)-(LX+7,LY+23),B,
PSET
```

Para movimentar o caminhão, faça estas modificações:

```
110 DIM A(6),B(6)
120 GET (0,0)-(23,7),A,G
160 PUT (X,Y)-(X+23,Y+7),A,PSET
200 IF PEEK(342)=253 AND Y<182
THEN Y=Y+2:GOTO 240
220 IF PEEK(338)=247 AND X<229
THEN X=X+3:GOTO 240
240 PUT (LX,LY)-(LX+23,LY+7),B,
PSET
```

COMO USAR DADOS EM BINÁRIO

Você pode colocar números em binário nas linhas **DATA**, se quiser, mas certifique-se antes de que cada número consiste de um byte de oito bits. Você terá, também, que adicionar estas linhas ao programa:

```
71 LET N=0
72 FOR J=1 TO 8
74 IF MID$(N$,J,1)="1" THEN N=N
+2^(8-J)
76 NEXT J

80 POKE L*32+1536+F,N
```

As novas linhas examinam os bytes bit por bit, transformando-os em números decimais. Estes são, resumidamente, equivalentes aos da tabela de conversão mostrada na introdução deste artigo.

A movimentação de figuras em alta

resolução será apresentada em detalhes num artigo próximo.

No Apple, entretanto, a geração de figuras em alta resolução não segue os mesmos padrões de outros computadores, sendo muito mais complicada (em artigo anterior, publicamos um editor de figuras móveis que permite aos usuários desse microcomputador utilizarem figuras compostas por blocos de 8x8 pontos).

Neste artigo você poderá treinar a utilização do editor de figuras. Os programas a seguir só funcionarão se as figuras correspondentes tiverem sido geradas pelo editor.

Carregue o programa editor de figuras no seu computador e coloque o desenho do pequeno monstro a seguir em linhas **DATA** usando asteriscos.

Que sistema de numeração — binário, decimal ou hexa — é melhor usar nas linhas *DATA*?

Dependendo das características do seu computador, pode ser melhor usar o sistema binário. Além de permitir a visualização do padrão do desenho, evitando contas, o sistema binário possibilita alterações de pequenas partes da figura.

Se você quiser empregar o mesmo padrão em outros programas, vale a pena converter os números para decimal ou hexa, que são mais compactos.

A utilização de números hexadecimais vai ajudá-los a se familiarizar com esse sistema, que é a base da programação em código de máquina.

Execute o programa editor e a figura será armazenada na memória. Feito isso, digite **NEW** e o seguinte programa:

```
10  HOME : HGR
20  SCALE= 1: ROT= 0
30 X = 140:Y = 90
40  GOTO 200
50 LX = X:LY = Y: GET A$
60 A =  ASC (A$)
70  IF A = 80 AND Y > 8 THEN Y
 = Y - 4: GOTO 200
80  IF A = 76 AND Y < 150 THEN
Y = Y + 4: GOTO 200
90  IF A = 90 AND X > 8 THEN X
 = X - 4: GOTO 200
100  IF A = 88 AND X < 270 THEN
 X = X + 4: GOTO 200
110  GOTO 50
200  HCOLOR= 0
210  DRAW 1 AT LX,LY
220  HCOLOR= 3
230  DRAW 1 AT X,Y
240  GOTO 50
```

Não há necessidade de copiar as linhas **DATA** geradas pelo programa editor. Uma vez criada, a figura permanecerá protegida no topo da memória até que o computador seja desligado, ou até que o programa editor crie outra forma. Nem mesmo **NEW** pode destruí-la.

O programa movimenta a figura com o auxílio das teclas P, L, X e Z. As linhas 50 a 110 fazem isso da maneira usual.

As linhas de 10 a 40 cuidam das condições iniciais: limpeza e definição da tela, escala, orientação e posição inicial da figura.

As linhas de 200 a 240 apagam o monstro de sua última posição, dada por LX e LY, e o desenham na nova, dada por X e Y.

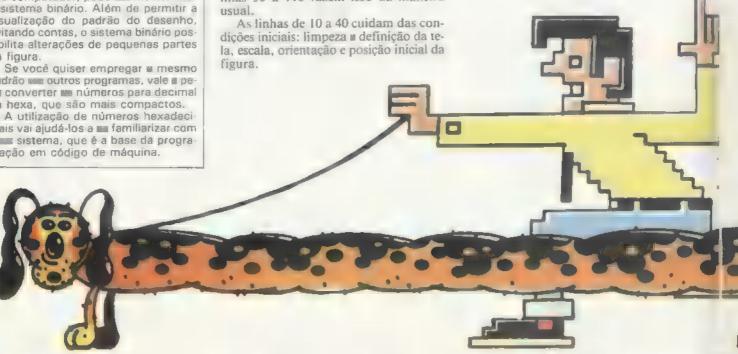
DESENHOS MAIS LARGOS

Desenhos compostos de mais de um bloco de 8x8 pontos exigem um cuidado especial com as coordenadas. Esse cuidado é necessário para que as figuras se posicionem de acordo com o planejado. Para familiarizar-se com esse tipo de desenho, gere os dois blocos que compõem o cachorro com o auxílio do programa editor.

Feito isso, digite **NEW** e carregue o programa anterior. Faça as seguintes modificações para mover o cachorro:

```
100  IF A = 88 AND X < 260 THEN
 X = X + 4: GOTO 200
215  DRAW 2 AT LX + 8,LY
235  DRAW 2 AT X + 8,Y
```

A condição após **AND** na linha 100 evita que o desenho ultrapasse a margem direita da tela. Ela teve que ser modificada porque agora o desenho é mais largo.

Como a figura tem dois blocos de 8x8

pontos, foi necessário colocar duas novas linhas com comandos **DRAW**, uma para desenhar e outra para apagar o segundo bloco.

É importante notar como são desenhados lado a lado os blocos que compõem o cachorro, ou seja, da esquerda para a direita, e de cima para baixo. Assim, as coordenadas no comando **DRAW** se referem ao canto superior esquerdo de cada bloco. Se o primeiro bloco está desenhado nas coordenadas X, Y, e queremos que o segundo seja colocado à sua direita, a posição deste último deve ser X + 8,Y. Isso acontece porque o canto superior esquerdo do segundo bloco deve ficar na mesma linha do canto superior esquerdo do primeiro (mesma coordenada vertical Y) oito pontos mais à direita.

UMA FIGURA ALTA E [illegible]

Continuando nosso treino, carregue novamente o editor e gere os dois blocos que desenham a silhueta de um homem.

Digite **NEW** e carregue o programa que movimenta o monstro. Faça então as seguintes modificações para mover o homem.

```
80  IF [illegible] = 76 AND Y < 140 THEN
Y = Y + 4: GOTO 200
215  DRAW 2 AT LX,LY + [illegible]
235  [illegible] 2 AT X,Y + 8
```

Da mesma maneira que no exemplo anterior, a condição na linha 80 teve que ser modificada para que o desenho, agora mais alto, não ultrapasse a margem inferior da tela.

As linhas 215 e 235 apagam e desenham, respectivamente, o segundo bloco. Se quisermos colocá-lo abaixo do primeiro bloco, teremos que situá-lo na mesma coluna, oito pontos mais embaixo. Assim, se a posição do primeiro bloco for X, Y, a do segundo deverá ser X,Y + 8.

[illegible]

Embora o MSX possua caracteres definíveis pelo usuário (com 8x8 pontos, como nas outras máquinas), geralmente é preferível utilizar os sprites, que são mais versáteis e mais adequados à movimentação de figuras.

Todavia, existem situações em que esses caracteres podem ser muito úteis. Uma aplicação muito comum é a redefinição total ou parcial do conjunto de caracteres. Isso às vezes é desejável em programas que devem utilizar alfabetos estrangeiros ou caracteres gráficos diferentes.

O conjunto de caracteres do MSX é muito completo, contendo todo o alfabeto grego e uma infinidade de outros caracteres gráficos. Falta-lhe, porém, a letra *è* (com acento grave), utilizada na língua francesa.

Para criar esse caractere, aplique o mesmo método já descrito neste artigo. A tabela que se segue traz a codificação do padrão do caractere desejado, pronta para ser colocada em uma linha **DATA**.

Binário	Hex	Decimal
00010000	10	16
00001000	[illegible]	8
00111100	3C	60
01100110	66	102
01111110	7E	126
01100000	60	96
00111100	3C	60
00000000	00	[illegible]

O formato decimal é mais fácil de usar. Mas mais difícil de calcular. O MSX aceita números binários e hexadecimais, desde que certos padrões sejam respeitados.

Para incorporar o novo caractere, use o programa:

```
10 SCREEN 0:KEY OFF
20 FOR I=0 TO 959
30 VPOKE BASE(0)+I,205
40 NEXT
```

```
50 E=205*8
60 FORI=E TO E+7
70 READ B$
80 VPOKE BASE(2)+I,VAL("&B"+B$)
90 NEXT I
100 FOR I=1 TO 2000:NEXT I
110 CLS:END
500 DATA 00010000
501 DATA 00001000
502 DATA 00111100
503 DATA 01100110
504 DATA 01111110
505 DATA 01100000
506 DATA 00111100
507 DATA 00000000
```

Ao ser executado o programa, a tela é ocupada por um símbolo gráfico. Em seguida esse símbolo se transforma no caractere que procuramos, ou seja, a letre *è* com acento grave. Após alguns instantes, a tela fica novamente limpa. O novo caractere, contudo, continua na memória e pode ser obtido da mesma maneira que o símbolo gráfico do qual tomou o lugar. Para isso, basta pressionar as teclas **GRAPH** e "E" ao mesmo tempo. O caractere só desaparecerá da memória quando utilizarmos o comando **SCREEN** ou quando desligarmos o computador.

ENTENDA O PROGRAMA

A linha 10 escolhe a tela de textos com 40 colunas e apaga do rodapé da tela as teclas de função. O laço das linhas 20 a 40 enche a tela com o caractere de código igual a 205 — correspondente a **GRAPH + E**.

Os padrões de todo o conjunto de caracteres estão guardados na VRAM — memória de vídeo — a partir do endereço dado por **BASE (2)**. Para saber como é o formato de um caractere, o computador multiplica o valor de seu código por 8 e soma esse valor ao de **BASE (2)**. O número que resultar dessa conta será o endereço do primeiro de uma série de oito bytes onde ficará armazenada a forma do caractere.

A linha 50 multiplica por 8 o código do caractere que vai ser modificado, para saber onde seu formato está na VRAM. O laço das linhas 70 a 90 coloca nesse local o padrão do novo caractere; os números são lidos com **READ** nas linhas **DATA**. Na linha 80, **VAL (&B + B$)** permite que sejam utilizados números binários.

A linha 100 retarda o processo até que a linha 110 apague a tela e termine o programa.

Se for preciso economizar espaço, as linhas **DATA** podem ser substituídas por uma única linha.

```
500 DATA 00010000,00001000,0011
1100,01100110,01111110,01100000
,00111100,00000000
```

Essa linha única, porém, não permite que se visualize claramente o caractere.

Se quisermos usar números hexadecimais, devemos fazer as seguintes modificações:

```
80 VPOKE BASE(2)+I,VAL("&H"+B$)
500 DATA 10,08,3C,66,7E,60,3C,0
0
```

Uma linha **DATA** desse tipo é bem mais fácil de ser digitada.

Para criar o mesmo caractere na tela de textos com 32 colunas, faça as seguintes modificações:

```
20 FOR I=0 TO 767
30 VPOKE BASE(5)+I,205
80 VPOKE BASE(7)+I,VAL("&H"+B$)
```

Note que a linha **DATA** deve estar em hexa.

PROGRAMAS EDITORES

Crie suas próprias figuras no Apple e no MSX, utilizando os programas editores fornecidos por *INPUT* para estes computadores. Eles calcularão os números nas linhas **DATA** para você.

Lembre-se de que não será preciso digitar essas linhas. Ambos os computadores possuem facilidades de edição que permitem usar a própria tela produzida pelo editor — uma vez interrompido o programa — para criar uma linha **DATA** com um número alto na frente (a partir de 5000).

No MSX isto é feito com as teclas do cursor, **INS**, **DEL** e **ENTER**. No Apple, com **ESC**, **I**, **J**, **K**, **M**, setas e **RETURN**.

No Apple e no TK-2000 pode-se gravar a tabela de figuras usando o comando **W** do monitor. Quem tiver disco, pode usar o comando **BSAVE**.

S

O Spectrum aceita dados tanto em binário quanto em decimal, mas não em hexa. Para ver como ele trabalha com gráficos, digite:

```
PRINT "(graphics A)"
```

Para poder obter o símbolo (graphics A), você deverá pressionar <**CAPS SHIFT**> e <**GRAPHICS**> ao mesmo tempo e depois teclar "A".

O que você vê parece um A maiúsculo normal. Mas você pode defini-lo como uma forma de 8x8 pontos.

Para fazer isto, tudo o que você precisa é de um programa de cinco linhas. Assim, se você quiser plantar o pinheiro da ilustração, deve digitar as linhas a seguir:

```
10 FOR n=0 TO 7
20 READ data
30 DATA BIN 00010000,BIN 0001
1000,BIN 00111000,BIN 0011110
0,BIN 01111100,BIN 01111110,
BIN 11111110,BIN 00010000
40 POKE USR "a"+n,data
50 NEXT n
```

Esse programa usa um laço **FOR...NEXT** nas linhas 10 e 50 para dar entrada às oito linhas do desenho em seqüência. A linha 20 diz ao computador para ler dados na linha **DATA** e a linha 40 coloca esses dados na memória do computador, usando o comando **POKE**.

Execute o programa e digite:

```
PRINT "(graphics A)"
```

Você verá que o "A" desapareceu e em seu lugar aparece um pinheiro. Se você digitar **NEW** agora, o programa será apagado da memória, mas o desenho ficará na memória até que o computador seja desligado. Assim, você pode movê-lo pela tela ou usá-lo para efeitos decorativos como se ele fosse um caractere qualquer. Tente este exemplo:

```
 5 CLS
10 FOR y=3 TO 19
20 LET x=INT (RND*20)+5
30 PRINT AT y,x; INK 4;"(grap
hics A)"
40 LET xx=INT (RND*20)+5
50 PRINT AT y,xx; INK 2;"(gra
phics A)"
60 NEXT y
```

Esses pinheiros poderiam ser usados como obstáculos para um jogo de esqui.

DESENHE UM NAVIO

Quando você quiser criar gráficos maiores que 8x8, desenhe dois ou mais blocos de 8x8.

O programa a seguir, por exemplo, cria a proa de um destróier (que você pode colocar em um de seus jogos, usando as técnicas apresentadas em *Programação de Jogos*):

```
10 FOR n=0 TO 7
20 READ [illegible]
30 [illegible] BIN 0,BIN 0,BIN 0,BIN
00000111,BIN 00000011,BIN 111
11111,BIN 00000011,BIN 111111
11,BIN 01111111,BIN 00111111
[illegible] POKE USR "a"+n,a
50 NEXT n
```

Dois aspectos devem [illegible] ressaltados. Primeiro: você pode usar **BIN 0** se a linha inteira for vazia. Segundo: coloque na linha 20 uma variável simples "a" em lugar de "data", que foi inserida [illegible] programa anterior para que este ficasse mais compreensível. Poderia ser qualquer outra coisa, como "b", "c" ou "x", ou mesmo "maria"; [illegible] para is[illegible] deveria ser utilizada uma variável idêntica na linha 40.

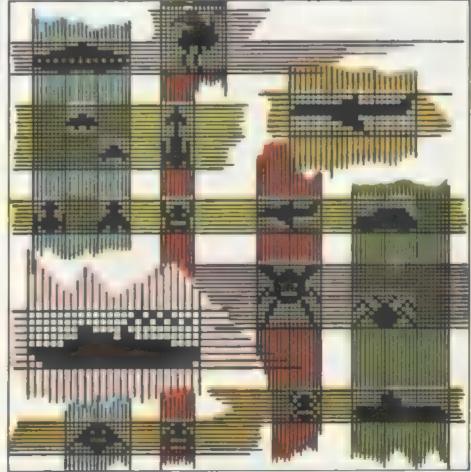
Quando você for entrar [illegible] dados pa[illegible] as outras duas partes do navio, não haverá necessidade de redigitar o programa todo. Depois de executá-lo a primeira vez, e com o primeiro gráfico na memória, mude a linha 40 para **USR "b"**, por exemplo. Mude também [illegible] linha 30 de modo que ela passe a conter os novos dados das outras partes do navio.

Tome cuidado para que [illegible] [illegible] linhas **DATA** tenham sempre oito dados, mesmo que estes sejam apenas zeros. Um número menor de linhas provocará um erro do tipo: E Out of DATA, 20:1. Coloque linhas [illegible] mais e você descobrirá que seu navio está afundando.

Finalmente, para entrar [illegible] dados em decimal, converta os binários de acordo com o método já descrito; [illegible] seguir, coloque-os [illegible] linha **DATA** omitindo a palavra **BIN**. A linha 30 do programa do destróier ficará assim:

```
30 DATA 0,0,0,7,3,255,127,63
```

E AGORA...

Uma vez compreendidos [illegible] princípios gerais, você pode colocar [illegible] computador qualquer um dos gráficos desta página. Depois de ter feito uns três deles, você verá que já tem prática suficiente para criar os seus próprios desenhos e colocá-los [illegible] computador.

UM ASSISTENTE DE ARTE

Mesmo não sendo um Leonardo da Vinci, você pode realizar belos desenhos com ■ ajuda do computador. Para isso, basta executar o programa apresentado nesta lição.

Você não precisa ser um Rafael ou mesmo um Pablo Picasso para desenhar com desenvoltura. Devidamente programado, seu computador pode se transformar em uma poderosa ferramenta de desenho, capacitando-o a realizar trabalhos gráficos de boa qualidade com liberdade igual (ou até maior) que ■ de um artista que trabalha apenas com material convencional: lápis e papel.

Na indústria, a combinação de programas de Desenho Auxiliado por Computador (conhecido pela sigla CAD, da expressão em língua inglesa *Computer Aided Design*) com computadores de grande porte permite não só ■ execução de desenhos de objetos com muita riqueza de detalhes como também a simulação de situações em que tais objetos são submetidos à ação de determinados fatores como carga, vento, vibração ou variações térmicas. Essas técnicas extrapolam a capacidade do micro doméstico, visto que exigem o processamento de um volume muito grande de dados, assim como uma capacidade gráfica altamente desenvolvida, para que ■ possa produzir imagens de vários ângulos do objeto em estudo.

Freqüentemente, no entanto, o desenhista necessita apenas estudar um detalhe ou a aparência geral do objeto. Nestes casos, o CAD se torna uma ferramenta capaz de substituir lápis e papel na execução de desenhos técnicos, permitindo o traçado preciso de curvas, linhas retas e figuras geométricas, com a vantagem da possibilidade de correção instantânea.

VANTAGENS DO PROGRAMA

O CAD pode ser aplicado aos computadores domésticos, seja para explorar o aspecto técnico do desenho, seja para produzir belas formas. Mas, apesar de quase todos os micros existentes no mercado brasileiro (com exceção das linhas Apple e TRS-80) já apresentarem comandos para desenhar formas elementares dentro do BASIC, é sempre necessário um longo programa para se construir formas mais sofisticadas. Exemplos de como isto é feito já foram apresentados em artigos anteriores.

A beleza do programa editor de desenhos consiste em que, uma vez na memória do computador, ele possibilita a criação de imagens até o limite gráfico da máquina, sem que seja necessário elaborar novos programas. Além disso, ele coloca todos os comandos para gráficos sob o controle direto do teclado; dessa forma, tudo o que se tem a fazer é escolher uma opção ou mover o cursor usando as teclas apropriadas. Esse recurso oferece, em alguns casos, possibilidades que não são encontradas no BASIC padrão do computador.

Como ■ habilidades gráficas dos diversos micros variam muito, existem grandes diferenças entre os programas, esquematizados para explorar ao máximo as vantagens ■ minimizar as fraquezas do computador ao qual estão destinados. Os Spectrum, por exemplo, podem mostrar facilmente texto ■ gráficos na mesma tela, o que é impossível nos micros das linhas Apple e TRS-Color. Por outro lado, ■ podem guardar desenhos que não estão à vista, enquanto o MSX, por exemplo, não pode.

Devido à limitada capacidade gráfica do ZX-81, não apresentaremos programa para esse micro. O programa para o Sinclair Spectrum rodará somente em máquinas com um mínimo de 48K.

AUXÍLIO AO DESENHO

Apesar das diferenças, os programas funcionam de forma semelhante: o usuário é apresentado ■ ■ menu (ou lista) de opções de desenho, que oferece ■ formas como linhas, elipses, círculos ou retângulos. Ao selecionar uma delas, você pode construir formas muito interessantes, usando as teclas de controle do cursor para posicioná-lo na tela; ■ linhas resultantes serão precisas e regulares. E se um erro for cometido, alguns computadores lhe darão ■ oportunidade de correção.

Você encontrará também ■ opção de mudar a cor com que desenha ou, em alguns casos, de colorir uma determinada área. Quando terminar, pode escolher entre apagar tudo e recomeçar ou gravar a imagem de forma que ela possa ser recolocada na tela, mais tarde. Isto lhe possibilitará também carregar na memória (e alterá-lo) um desenho de seu agrado que não tenha sido montado por este programa.

EXECUTE O PROGRAMA

As diferenças entre as diversas máquinas serão explicadas nos blocos específicos. Em todos os casos, o programa aparece dividido em duas partes: nesta lição trataremos da primeira (desenhos com linhas simples); a segunda parte, com variações mais sofisticadas, será abordada no próximo artigo.

Um menu e o cursor serão mostrados

- DESENHE COM A AJUDA DO COMPUTADOR
- COMO ESTENDER OS COMANDOS GRÁFICOS JÁ EXISTENTES
- COMO USAR O PROGRAMA
- APRENDA A DESENHAR A MÃO LIVRE
- COMO OBTER MAIOR PRECISÃO USANDO A OPÇÃO "LINHA"
- APRENDA A USAR CORES

na tela, assim que o programa for executado. Para selecionar um item do menu, coloque o cursor à sua esquerda, usando as teclas Q (para cima), A (para baixo), O (para a esquerda) e P (para a direita). Em seguida, pressione <**ENTER**> para fazer a escolha; o vídeo será apagado, deixando o cursor numa tela vazia. Você poderá, a qualquer momento, retornar ao menu (ou deixá-lo) apenas pressionando <**ENTER**>.

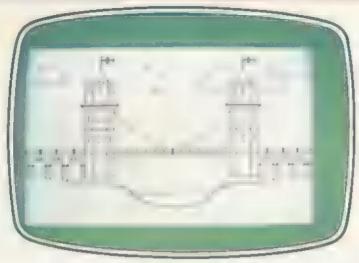
A primeira opção do menu é "Desenhar", que coloca pontos na tela. Para usar essa opção, posicione o cursor no ponto em que você deseja iniciar o desenho e pressione a barra de espaços. Ao ser movimentado, o cursor deixará uma trilha de pontos. O movimento pode ser acelerado pressionando-se a tecla <**SHIFT**> juntamente com a tecla que controla a direção do deslocamento. Para terminar a linha desenhada, acione a barra de espaços novamente, o que lhe permitirá mover o cursor sem marcar a tela ou retornar ao menu para selecionar outra opção.

A opção "Linha" funciona de forma semelhante à anterior. A diferença é que a linha obtida é cheia: pressione <**ENTER**> para selecioná-la e tecle a barra de espaços para desenhar e novamente para terminar. A opção selecionada permanecerá disponível até que uma nova seja escolhida. Dessa forma, pode-se ir até outro ponto e iniciar uma nova linha, e assim por diante.

Existem duas opções que lhe permitem colorir a tela: "Fundo", para pintar a moldura e o fundo e "Tinta", para colorir à medida que se desenha. Selecionada uma dessas alternativas, uma mensagem será mostrada para que você escolha as cores. O cursor lhe mostrará a cor em uso, de forma que se poderá ver o efeito instantaneamente.

Depois de ter respondido a todas as questões, o programa volta ao desenho, com o cursor no local onde foi deixado. Pode-se fazer a opção "Tinta" antes ou enquanto se desenha. Somente após se ter deixado a opção de desenho (ao pressionar a barra de espaços pela segunda vez) é que a linha será fixada.

Atenção, o programa contém código de máquina, de forma que deve ser gravado (**SAVE**) antes de se executar (**RUN**).

```
  10 BORDER 4: PAPER 7: INK 0:
  OVER 0: CLS
  20 POKE 23658,0: LET OP=1
  40 DIM O(12): FOR N=1 TO 12:
  READ O(N): NEXT N: LET x=127:
   LET y=87
  60 LET x1=x: LET y1=y
  65 LET xx=0: LET yy=0
  70 FOR n=65368 TO 65368+73:
  READ a: POKE n,a: NEXT n
 100 RAND USR 65380
1020 PRINT  INK 9;AT 2,9;"
       "
1030 PRINT  INK 9;AT 3,9;"   DE
SENHAR   "
1040 PRINT  INK 9;AT 4,9;"   LI
NHA      "
1050 PRINT  INK 9;AT 5,9;"   FU
NDO      "
1060 PRINT  INK 9;AT 6,9;"   TI
NTA      "
1070 PRINT  INK 9;AT 7,9;"   RE
TANGULO  "
1080 PRINT  INK 9;AT 8,9;"   CA
IXA      "
1090 PRINT  INK 9;AT 9,9;"   CI
RCULO    "
1100 PRINT  INK 9;AT 10,9;"    A
PAGAR     "
1110 PRINT  INK 9;AT 11,9;"    O
OPS       "
1115 PRINT  INK 9;AT 12,9;"    C
OPIAR     "
1120 PRINT  INK 9;AT 13,9;"    C
ARREGAR   "
1130 PRINT  INK 9;AT 14,9;"    G
RAVAR     "
1140 PRINT  INK 9;AT 15,9;"
          "
1150 PLOT 72,48: DRAW  INK 9;11
1,0: DRAW  INK 9;0,111: DRAW  I
NK 9;-111,0: DRAW  INK 9;0,-111
1155 FOR n=1 TO 100: NEXT n
1160 PRINT  INVERSE 1; PAPER 9;.
```

Desenhos a traços, como ■ mostrados na tela do Spectrum... ... podem ser realizados facilmente ■ o programa deste artigo.

```
AT OP+2,10;">"
1170 PAUSE 0: LET AS-INKEYS: IF
 AS="" THEN  GOTO 1170
1175 IF AS-CHRS 13 AND OP-9 THE
N  RAND USR 65404: RAND USR 653
68: GOTO 1000
1180 IF AS-CHRS 13 THEN  RAND U
■ 65392: RAND USR 65368: FOR n
=1 TO 100: NEXT n: GOTO O(OP)
1190 PRINT AT OP+2,10;" "
1200 IF AS-"a" THEN  LET OP-OP+
1: IF OP=13 THEN  LET OP-12
1210 IF AS-"q" THEN  LET OP-OP-
1: IF OP-0 ■  LET OP=1
1220 GOTO 1160
2000 REM desenhar
2005 FOR n-1 TO 50: NEXT n
2010 GOSUB 8000
2060 IF INKEYS-CHRS 13 THEN  RA
ND USR 65380: GOTO 1000
2070 IF INKEYS<>CHRS 32 THEN  G
OTO 2010
2080 FOR n-1 TO 50: NEXT n
2090 GOSUB 8000: PLOT x,y
2095 IF INKEYS-CHRS 13 THEN  RA
ND USR 65380: GOTO 1000
2100 IF INKEYS-CHRS 32 THEN  GO
TO 2005
2110 GOTO 2090
2500 REM linha
2505 FOR n-1 TO 50: NEXT n
2510 GOSUB 8000
2515 IF INKEYS-CHRS 13 THEN  RA
ND USR 65380: GOTO 1000
2520 IF INKEYS<>CHRS 32 THEN  G
OTO 2510
2525 FOR n-1 TO 50: NEXT n
2530 LET xx=0: LET yy=0: LET hx
=x: LET hy-y
2540 GOSUB 8000: PLOT hx,hy: DR
AW  OVER 1;xx,yy: FOR n-1 TO 5:
 NEXT n: PLOT hx,hy: DRAW  OVER
 1;xx,yy
2550 IF INKEYS<>CHRS 32 THEN  G
OTO 2540
2560 PLOT hx,hy: DRAW xx,yy: GO
TO 2500
3000 REM fundo ■ borda
3010 PRINT #1;AT 0,0;"Cor de fu
ndo (0 a 7)?"
3015 LET a$-INKEY$
3020 IF a$<"0" OR a$>"7" THEN
GOTO 3015
3030 POKE 65535,VAL a$*8
3035 RAND USR 65416
3040 FOR n-1 TO 50: NEXT n
3050 PRINT #1;AT 0,0;"Cor da bo
rda (0 a 7)?"
3060 LET a$-INKEY$
3070 IF a$<"0" OR a$>"7" THEN
GOTO 3060
3080 BORDER VAL a$
3090 PRINT #1;AT 0,0;" ";TAB 31
;"  ";TAB 31;" "
3095 RAND USR 65416: RAND USR 6
5380: GOTO 1000
3500 REM tinta
3510 PRINT #1;AT 0,0;"Selecione
 tinta (0 ■ 7)"
3520 LET a$-INKEY$: IF a$<"0" O
R a$>"7" THEN  GOTO 3520
3530 INK VAL a$
3540 PRINT #1;AT 0,0;" ";TAB 31
;" ": GOTO 1000
7500 GOTO 1000
8000 ■ rotina INKEYS
8005 PLOT  OVER 1;x,y
8010 LET A$-INKEYS
8020 IF A$-"q" AND y<175 THEN
LET y1-y+1: LET yy-yy+1
8030 IF A$-"a" AND y>0 THEN  LE
T y1-y-1: LET yy=yy-1
8040 IF A$-"p" AND x<255 THEN
LET x1-x+1: LET xx-xx+1
8050 IF A$-"o" ■ x>0 THEN  LE
T x1-x-1: LET xx-xx-1
8060 IF A$-"Q" AND y<172 THEN
LET y1-y+4: LET yy-yy+4
8070 IF a$="A" AND y>3 THEN  LE
T y1-y-4: LET yy=yy-4
8080 IF a$-"P" AND x<252 THEN
LET x1-x+4: LET xx-xx+4
8090 IF a$="O" AND x>3 THEN  LE
T x1-x-4: LET xx-xx-4
8095 PLOT  OVER 1;x,y
8100 LET x=x1: LET y=y1: RETURN

9000 DATA 2000,2500,3000,3500,4
000,4020,5000,5500,0,6000,7000,
7500
9010 DATA 17,0,64,33,80,195,1,0
,27,237,176,201,17,80,195,33,0,
64,1,0,27,237,176,201
9020 DATA 17,168,222,33,80,195,
1,0,27,237,176,201,17,80,195,33
,168,222,1,0,27,237,176,201
9030 DATA 33,0,88,6,4,197,6,176
,203,158,203,166,203,174,58,255
,255,134,119,35,16,242,193,16,2
36,201
```

T

O programa começa mostrando um submenu, que permite escolher valores para o modo, tela e cor. Use as setas para mover o cursor e a barra de espaços para fazer a seleção.

Depois de escolher ■ valores apropriados, pressione <**ENTER**> para ter o menu principal. Para acessar as opções, posicione o cursor (usando as setas) ao lado do número do item ■ pressione a barra de espaços. Isso proporciona a limpeza da tela, deixando-a com o cursor pronto para desenhar. Para retornar ■ menu ■ qualquer momento, pressione <**ENTER**>.

A primeira opção do menu principal é "Mudar a tela", que dá acesso ao submenu. A opção seguinte é "Desenhar", usada para desenhar ■ mão livre, como ■ faz com um lápis. Quando selecionada, essa opção permite mover o cursor até o ponto escolhido para dar começo ao desenho; enquanto a barra de espaços for mantida pressionada, uma linha permanecerá na tela. Para interromper o desenho, basta soltá-la. Pode-se acelerar o movimento pressionando-se <**CLEAR**>. Observe que isso só funciona quando não se desenha.

Na próxima lição, você verá como adicionar cor ao desenho... ... como nestes exemplos mostrados na tela do TRS-Color.

Em qualquer das opções de desenho, pode-se mudar de cor pressionando-se um número entre 0 a 8: isto lhe fornecerá a cor correspondente que consta do seu manual. Para sair de "Desenhar" pressione **<ENTER>**, voltando ao menu principal.

A opção "Traçar Linha" permite desenhar linhas retas. Selecione a opção com as setas, desloque o cursor até o ponto onde deseja iniciar o desenho e pressione a barra de espaços. Escolha então a cor (números de 0 a 8). A cor do cursor depende da cor de fundo, mas não afeta a cor da linha. Mova o cursor até o ponto onde deseja terminar a linha e acione a barra de espaços para desenhá-la. Pressione **<ENTER>** para voltar ao menu principal. Tecle **<CLEAR>** para deixar o programa.

```
10 PCLEAR 8
20 CLS:PA=239:PB=223:PC=247:PD=
191:PE=183
30 DIM SC(1228),CP(614)
40 DEFFNR(Z)=SQR((XS-X)*(XS-X)+
(YS-Y)*(YS-Y))
50 MD=-1:ST=-1:OP=1:CL=1:X=127:
Y=95
60 GOTO 80
70 FOR K=1 TO 1500:NEXT
80 GOSUB 1000
90 IF MD<0 THEN PRINT @449,"err
o NAO FOI DEFINIDO MODO       GR
AFICO":GOTO 70
100 IF ST<0 THEN PRINT @449,"er
ro NAO FOI DEFINIDO TIPO DE    T
ELA":GOTO 70
110 OT=1:PMODE MD,1
120 CLS:PRINT @7,"MENU PRINCIPA
L"
130 PRINT @103,"1- MUDAR TELA":
PRINT @135,"2- DESENHAR":PRINT
@167,"3- TRACAR LINHA":PRINT @1
99,"4- RETANGULO":PRINT @231,"5
- CAIXA":PRINT @263,"6- CIRCULO
140 PRINT @295,"7- DISCO":PRINT
 @327,"8- ELIPSE":PRINT @359,"9
- COPIAR":PRINT @390,"10- PREEN
CHER":PRINT @422,"11- ERRO":PRI
NT @454,"12- GRAVAR/CARREGAR"
150 POKE 1097+OT*32,128
160 IF PEEK(341)=PC AND OT>1 TH
EN POKE 1097+OT*32,96:OT=OT-1:G
OTO 150
170 IF PEEK(342)=PC AND OT<12 T
HEN POKE 1097+OT*32,96:OT=OT+1:
GOTO 150
180 IF PEEK(345)=PC THEN 200
190 IF PEEK(339)=PD THEN CLS:EN
D ELSE 160
200 IF OT<>11 THEN 220
210 PMODE MD,5:PUT (0,0)-(255,1
91),SC:GOSUB 500:PMODE MD,1:GOT
O 120
220 GOSUB 510
230 GET(0,0)-(255,191),SC
240 EF=0
250 ON OT GOSUB 1000,2000,3000,
3000,3000,3000,3000,3000,4000,5
000,0,6000
260 GOTO 120
500 FOR K=1 TO 4:PCOPY K+4 TO K
:NEXT:RETURN
510 FOR K=5 TO 8:PCOPY K-4 TO K
:NEXT:RETURN
1000 CLS:PRINT @6,"preparacao d
a tela"
1010 PRINT @102,"1- MODO GRAFIC
O":PRINT @166,"2- TIPO DE TELA"
:PRINT @230,"3- LIMPAR TELA"
1020 PK=PEEK(1062+64*OP):POKE 1
062+64*OP,63 AND PK
1030 IF PEEK(341)=PC AND OP>1 T
HEN POKE 1062+64*OP,PK:OP=OP-1:
GOTO 1020
1040 IF PEEK(342)=PC AND OP<3 T
HEN POKE 1062+64*OP,PK:OP=OP+1:
GOTO 1020
1050 IF PEEK(338)=PD THEN RETUR
N
1060 IF PEEK(345)=PC THEN 1080
1070 GOTO 1030
1080 CLS:ON OP GOSUB 1200,1300,
1400
1090 FOR K=1 TO 200:NEXT:GOTO 1
000
1200 PRINT @33,"QUE MODO GRAFIC
O DESEJA USAR     (0-4) ?";
1210 A$=INKEY$:IF A$<"0" OR A$>
"4" THEN 1210
1220 PRINT A$:MD=VAL(A$):PMODE
MD,1:RETURN
1300 IF MD<0 THEN PRINT @449,"e
rro NAO FOI DEFINIDO MODO      G
RAFICO":FOR K=1 TO 1000:NEXT:RE
TURN
1310 PRINT @33,"QUE TIPO DE TEL
A DESEJA USAR     (0 OU 1) ?";
1320 A$=INKEY$:IF A$<"0" OR A$>
"1" THEN 1320
1330 PRINT A$:ST=VAL(A$):RETURN
1400 PRINT @33,"TEM CERTEZA QUE
 QUER LIMPAR A   TELA (S/N)?"
1410 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 1410
1420 IF A$="N" THEN RETURN
1430 PRINT @129,"COM QUE COR VO
CE DESEJA LIMPAR  A TELA (0-8)?
";
1440 A$=INKEY$:IF A$<"0" OR A$>
"8" THEN 1440
1450 PRINT A$:PMODE MD,5:PCLS V
AL(A$):PMODE MD,1:PCLS VAL(A$):
RETURN
1500 FOR K=0 TO 7:IF PEEK(338+K
)=PA THEN CL=K
1510 NEXT:IF PEEK (338)=PB THEN
 CL=8
1520 IF PEEK(338)=PD THEN EF=1:
RETURN
1530 DRAW"BM"+STR$(X)+","+STR$(
Y)+";C"+STR$((PPOINT(X,Y)+3)AND
 7)+"BE4G2BD4NF2BL4NG2BU4H2":CO
LOR CL
1550 IF PEEK(339)=PD THEN DF=10
 ELSE DF=1
1560 IF PEEK(341)=PC THEN Y=Y-D
F:GOTO 1610
1570 IF PEEK(342)=PC THEN Y=Y+D
F:GOTO 1610
1580 IF PEEK(343)=PC THEN X=X-D
F:GOTO 1610
1590 IF PEEK(344)=PC THEN X=X+D
```

```
F:GOTO 1610
1600 RETURN
1610 X=255 AND ■
1620 IF Y>191 OR Y<0 THEN Y=Y+1
91*(2*(Y>191)+1)
1630 GOSUB 500:RETURN
2000 SCREEN 1,ST:GOSUB 1500
2010 IF EF=1 GOSUB 500:RETURN
2020 IF PEEK(345)=PC THEN ■■■■
 MD,5:PSET(X,Y,CL):PMODE MD,1:P
OKE 345,255
2030 GOTO 2000
3000 SCREEN 1,ST:GOSUB 1500
3010 IF EF=1 GOSUB 500:RETURN
3020 IF PEEK(345)<>PC THEN 3000
3030 XS=X:YS=Y
3040 GOSUB 1500
3050 GOSUB 500
3060 IF EF=1 THEN RETURN
3070 ON OT GOSUB 0,0,3130,3140,
3140,3150,3150,3160,0,0,0
3080 IF PEEK(345)=PC THEN 3100
3090 GOTO 3040
3100 IF OT=5 ■■■■ LINE(X,Y)-(XS
,YS),PSET,BF
3110 IF OT=7 THEN PAINT (X,Y),C
L,CL
3120 ■■■■ 510:GOTO 3000
3130 LINE(XS,YS)-(X,Y),PSET:RET
URN
3140 LINE(XS,YS)-(X,Y),PSET,B:R
ETURN
3150 CIRCLE(X,Y),FNR(Z),CL:RETU
RN
3160 IF XS<>X THEN 3190
3170 IF(2*YS-Y)>191 OR (2*YS-Y)
<0 THEN RETURN
3180 LINE (X,Y)-(X,2*YS-Y),PSET
:RETURN
3190 CIRCLE(XS,YS),ABS(X-XS),CL
,ABS((Y-YS)/(X-XS)):RETURN
4000 RETURN
5000 RETURN
6000 RETURN
```

O programa começa oferecendo ■ possibilidade de trabalhar nas páginas 1 ou 2 de gráficos de alta resolução. A seguir, escolhe-se a cor de fundo.

Depois que ■ tela é preenchida com a cor de sua preferência, chega-se ■■ menu principal de opções. Tanto neste como no programa anterior, seleciona-se o item desejado pressionando-se as setas do teclado até que o vídeo comece a piscar. Então, pressiona-se **<CR>** ou **<RETURN>**.

Para desenhar linhas de qualquer tamanho e em qualquer posição faça a opção "TRAÇAR LINHAS" (isso é feito da maneira usual). Posicione então o cursor em uma das extremidades da linha que você deseja. Pressione ■ barra de espaços. Você ouvirá um bip e um ponto aparecerá na tela. Mova ■ cursor até a outra extremidade da linha e acione novamente ■ barra de espaços. Sua linha será traçada no mesmo instante.

Se você quiser desenhar em outras cores, pressione uma das teclas de 0 a 7. Cada uma delas lhe oferece a cor corresponde, cuja tabela você encontrará no seu manual. Nesse momento, você notará que ■ questão das cores não é muito simples. Existem várias interações que ocorrem entre as cores próximas. Assim, nem sempre a cor escolhida aparecerá adequadamente na tela. Logo, no entanto, você aprenderá a contornar esse problema.

Outra opção disponível é "APAGA", que limpa a tela em uso, dando a você a possibilidade de escolher uma nova cor de fundo.

O menu principal oferece nove alternativas, mas apenas algumas delas estão disponíveis no momento. A primeira ("NOVA TELA") leva você de volta ao início do programa, permitindo que se refaçam as opções. Note que a tela escolhida será apagada.

As cinco opções seguintes dizem respeito à possibilidade de desenhar. Por enquanto, porém, apenas "DESENHAR" e "TRAÇAR LINHAS" estão à sua disposição. A opção "DESENHAR" funciona da seguinte forma: ao ser feita ■ escolha do item, ele ficará piscando; pressione **<CR>** ou **<RETURN>**; imediatamente, a tela gráfica será colocada à sua disposição, com o cursor posicionado em seu centro. O cursor pode ser movimentado pela tela usando-se as teclas I, J, M, K, U, N, ', ', O. Para começar a desenhar, pressione a barra de espaços. Você ouvirá um bip ■ o cursor desaparecerá. A partir desse momento, todo o trajeto percorrido ficará marcado na tela. Para parar de desenhar e ter o cursor de volta, tecle novamente a barra de espaços.

Para voltar ao menu principal, pressione mais uma vez ■ comando **<CR>**.

A opção "MUDAR DE TELA" exige um cuidado especial. Só é possível desenhar ■■ tela escolhida no início do programa. Assim, essa opção permite apenas que você dê uma olhada na outra tela, que não está sendo usada. Depois de selecioná-la, escolha ■■■ das alternativas de desenho. A outra tela lhe será então mostrada; não se esqueça, porém, de que não é possível desenhar sobre ela. Para retornar à tela original, tecle **<CR>** e selecione novamente "MUDAR DE TELA".

Finalmente, aprenda a sair do programa, teclando **<ESC>** quando estiver no ■■■■ principal.

```
10  ONERR  GOTO 9000
20 PI = 4 *  ATN (1)
30  DIM CL$(7),OP$(8)
40  FOR I = 0 TO 7: READ CL$(I)
: NEXT
50  FOR I = 0 TO 8: READ OP$(I)
: NEXT
60  TEXT : HOME : PRINT "Escolh
a ■ pagina de alta resolucao:"
70  VTAB 6: INPUT "Pagina? (1 o
u 2) ";PG
80  IF PG <  > 1 AND PG <  > 2
THEN 60
90  HOME : PRINT "Escolha a cor
 de fundo:"
100  VTAB 10: FOR I = ■ TO 7: H
TAB 8: PRINT CL$(I): NEXT
110 I = 0:LM = PG * 8192
120  FLASH : VTAB 10 + I: HTAB
6: PRINT  SPC( 2);CL$(I); SPC(
1);
130  GET A$: IF A$ =  CHR$ (13)
 THEN 210
140  IF A$ <  >  CHR$ (8) AND A
$ <  >  CHR$ (21) THEN 130
150  NORMAL : VTAB 10 + I: HTAB
 6: PRINT  SPC( 2);CL$(I); SPC(
 2);
160  IF A$ =  CHR$ (8) THEN 190

170  IF I = 7 THEN I = 0: GOTO
120
180 I = I + 1: GOTO 120
190  IF I = 0 THEN I = 7: GOTO
120
200 I = ■ - 1: GOTO 120
210  IF PG = 1 THEN  HGR : GOTO
 230
220  HGR2
230  NORMAL : POKE  - 16304,0:
POKE  - 16302,0: POKE  - 16301
+ PG,0: POKE  - 16297,0: HCOLOR
= I: HPLOT 0,0: CALL 62454:X =
100:Y = X: HCOLOR= 3
240 OP = 0
250  FOR I = 1 TO 200: NEXT : T
EXT : HOME
260  PRINT "Escolha a sua opcao
:"
270  VTAB 10: FOR I = 0 TO 8: ■
TAB 8: PRINT OP$(I): NEXT
280 SP$ = "  "
290  FLASH : VTAB 10 + OP: HTAB
 6: PRINT SP$;OP$(OP);" ";
300  GET A$: IF A$ =  CHR$ (13)
 THEN  NORMAL : GOTO 390
310  IF A$ =  CHR$ (27) THEN  N
ORMAL : HOME : END
320  IF A$ <  >  CHR$ (8) AND A
$ <  >  CHR$ (21) THEN 300
330  NORMAL : HTAB 6: PRINT SP$
;OP$(OP);SP$;
340  IF A$ =  CHR$ (8) THEN 370

350  IF OP = 8 THEN OP = 0: GOT
■ 290
360 OP = OP + 1: GOTO 290
370  IF OP = 0 THEN OP = 8: GOT
O 290
380 OP = OP - 1: GOTO 290
390  POKE  - 16301 + PG,0: POKE
 - 16304,0:AA = 0
400  ON OP + 1 GOTO 60,2000,300
0,3000,3000,4000,5000,6000,7000

1000 X1 = X:Y1 = Y
```

```
1010  GET A$
1020  IF A$ > "0" AND A$ < "7"
THEN  HCOLOR=  VAL (A$): GOTO 1
010
1030  IF A$ =  CHR$ (13) THEN
POKE M,ME: TEXT : GOTO 290
1040  IF A$ = "I" THEN Y = Y1 -
 2
1050  IF A$ = "M" THEN Y = Y1 +
 2
1060  IF A$ = "J" THEN X = X1 -
 2
1070  IF A$ = "K" THEN X = X1 +
 2
1080  IF A$ = "O" THEN X = X1 +
 1:Y = Y1 - 1
1090  IF A$ = "U" THEN X = X1 -
 1:Y = Y1 - 1
1100  IF A$ = "N" THEN X = X1
 1:Y = Y1 + 1
1110  IF A$ = "," THEN X = X1 +
 1:Y = Y1 + 1
1120  RETURN
1500  IF M <  > 0 THEN  POKE M,
ME
1520 L =  INT (Y / 64):C =  INT
 (Y / 8) - (8 *  INT (Y / 64)):
T = Y - (8 *  INT (Y / 8))
1530 M = LM + (L * 40) + (C * 1
28) + (T * 1024)
1540 M = M + ( INT (X / 7))
1550 ME =  PEEK (M)
1560  POKE M,255 - 2 ^ (X - ( I
NT (X / 7) * 7))
1570  RETURN
2000  IF AA = 0 THEN  GOSUB 150
0
2010  GOSUB 1000: IF A$ = " " T
HEN AA =  ABS (AA - 1): IF AA =
 1 THEN  POKE M,ME
2020  IF AA = 1 THEN  HPLOT X,Y
 TO X1,Y1
2030  GOTO 2000
3000 CI = 0
3010  GOSUB 1500: GOSUB 1000: I
F A$ <  >  CHR$ (32) THEN  GOTO
 3010
3020  POKE M,ME
3030 CI = CI + 1:PX(CI) = X:PY(
CI) = Y: PRINT  CHR$ (7):  HPLO
T X,Y:M = 0
3040  IF CI <  > 2 THEN 3010
3050  ON OP - 1 GOTO 3060,3070,
3080
3060  HPLOT PX(1),PY(1) TO PX(2
),PY(2):M = 0: GOTO 3000
3070  PRINT  CHR$ (7);: PRINT
CHR$ (7);: GOTO 3000: REM  ROTI
NA NAO DISPONIVEL
3080  PRINT  CHR$ (7);: PRINT
CHR$ (7);: GOTO 3000: REM  ROTI
NA NAO DISPONIVEL
4000  PRINT  CHR$ (7);: PRINT
CHR$ (7);: GOTO 250: REM  ROTIN
A NAO DISPONIVEL
5000  IF PG = 1 THEN  HGR : GOT
O 5020
5010  HGR2
5020  TEXT : GOTO 80
6000 PG =  INT (PG - ( - 1 ^ PG
)): GOTO 250
7000  PRINT  CHR$ (7);: PRINT
CHR$ (7);: GOTO 250: REM  ROTIN
A NAO DISPONIVEL
9000  IF  PEEK (222) = 53 THEN
X = X1:Y = Y1: PRINT  CHR$ (7):
: RESUME
9010  PRINT  CHR$ (7);: GOTO 25
0
10000  DATA  PRETO 1,VERDE,VIOL
ETA,BRANCO,PRETO 2,LARANJA,AZUL
,BRANCO 2
10010  DATA  NOVA TELA,DESENHAR
,TRACAR LINHAS,RETANGULO,CIRCUL
O,ELIPSE,APAGA,MUDAR DE TELA,GR
AVAR/CARREGAR
```

O programa começa perguntando que cor de fundo você deseja. Responda com um número de 0 a 15. A correspondência de cores está no manual do seu micro. Em seguida, a tela gráfica de alta resolução será colocada à sua vista, mostrando no rodapé as opções disponíveis. Elas são doze. No entanto, apenas algumas estão disponíveis no momento. As outras serão apresentadas no próximo artigo.

Inicialmente, temos **Desenhar**, que permite traçar linhas em qualquer direção na tela. Selecione a opção colocando o quadrado sobre a abreviatura Des e tecle <**RETURN**>. Você verá então o pequeno cursor na tela. Ele pode ser movimentado com as setas do teclado. Pressione a barra de espaços no ponto em que quiser começar a desenhar: para onde quer que você movimente o cursor ele deixará um trilho marcado na cor de sua escolha.

Mas, como escolher a cor? É fácil: basta teclar, quando dentro de um módulo de desenho, uma tecla entre O e E, como se fosse uma numeração hexadecimal. Ou seja, os números de 0 até 9 funcionam normalmente; para escolher cores que tenham código de 11 a 15, use as letras de A a E. Para sair desse módulo e escolher outra coisa, pressione novamente <**RETURN**>; você obterá, assim, controle sobre o quadrado das opções.

A opção **Pintar** tem o efeito do comando **PAINT** do BASIC. Ela preencherá uma figura com a cor que estiver sendo usada. Deve-se tomar o cuidado de usar a mesma cor da periferia da figura para que o computador não estrague o seu desenho. Outra coisa importante é que a figura deve ser totalmente fechada; senão, o computador pintará toda a tela com a cor de sua preferência.

XXX provoca o apagamento total e definitivo do desenho. Portanto, tenha cuidado: se você pressionar <**RETURN**> com o quadrado sobre essa opção, o programa retornará ao início e apagará tudo o que foi desenhado.

```
10 X=100:Y=100:CO=15
20 PI=4*ATN(1)
30 SCREEN0:COLOR 15,4,4
40 CLS:LOCATE 5,5:PRINT"Escolha
 a cor de fundo da página  gráf
ica (0 a 15):"
50 INPUT CL:IF CL>15 OR CL<0 TH
EN PRINT"Valor ilegal!":GOTO 50
60  COLOR15,CL,CL:SCREEN2:OP=1
70  OPEN"GRP:"FOR OUTPUT AS #1
80  PSET(8,166),0
90  PRINT#1," Apa  Des  Lin  Re
t  Cai  Cir":PSET(8,176),0:PRIN
T#1," Dis  Eli  Pin  XXX"
100 CLOSE
110 GOSUB390:LINE (C,L)-(C+24,L
+9),15,B
120 A$=INKEY$:IFA$=""THEN120
130 GOSUB390
140 IFA$=CHR$(13)THEN200
150 LINE (C,L)-(C+24,L+9),CL,B
160 IFA$=CHR$(29)THENOP=OP-1:IF
OP<1THENOP=10
170 IFA$=CHR$(28)THENOP=OP+1:IF
OP>10THENOP=1
180 GOTO 110
190 AA=0
200 ON OP GOTO 480,440,480,480,
480,480,480,600,740,770
210 X1=X:Y1=Y
220 A$=INKEY$:IFA$=""THEN220
230 IFA$>"0"ANDA$<"9"THENCO=VAL
(A$)
240 IFA$>"A"ANDA$<"E"THENCO=ASC
(A$)-55
250 IFA$=CHR$(30)THENY=Y1-1
260 IFA$=CHR$(31)THENY=Y1+1
270 IFA$=CHR$(28)THENX=X1+1
280 IFA$=CHR$(29)THENX=X1-1
290 IFA$=CHR$(13)THENGOTO110
300 RETURN
310 IFM>0THENVPOKEBASE(12)+M+Y1
MOD8,ME:VPOKEBASE(11)+M+Y1MOD8,
MF
320 MX=INT(X/8):MY=INT(Y/8)
330 M=MY*32+MX:M=M*8
340 ME=VPEEK(BASE(12)+M+YMOD8)
350 MF=VPEEK(BASE(11)+M+YMOD8)
360 VPOKE BASE(12)+M+YMOD8,2^(7
-XMOD8)
370 VPOKE BASE(11)+M+YMOD8,(4+(
CL=4))*16
380 RETURN
390 IFOP<7THENC=(OP*5)*8ELSEC=(
OP-6)*5*8
400 C=C-26
410 IFOP<7THENL=164ELSE L=174:P
SET(C,L),0
420 RETURN
430 GOTO 200
440 IFAA=0THENGOSUB310
450 GOSUB210:IFA$=" "THENAA=ABS
(AA-1)
460 IFAA=1THENLINE(X,Y)-(X1,Y1)
,CO
470 GOTO440
480 GOTO 110
600 GOTO 110
740 GOSUB310:GOSUB210:IFA$<>CHR
$(32)THEN740
750 PAINT (X,Y),CO
760 GOTO 740
770 GOTO 30
```

ROTINAS PARA O CAD

- NOVAS POSSIBILIDADES DO PROGRAMA CAD
- OPÇÕES SOFISTICADAS
- CORRIGIR E COPIAR
- CORES NO SPECTRUM

Adicione rotinas especiais ao seu programa CAD (Desenho Assistido por Computador) e veja como tirar o melhor proveito da sofisticação que elas proporcionam à listagem original.

Com o programa CAD (Desenho Assistido por Computador), apresentado na página 414, vimos colocar os recursos gráficos do computador sob controle direto do teclado, o que nos permitiu fazer alguns desenhos bem sofisticados. Se quisermos, porém, explorar todo o potencial do programa — especialmente a possibilidade de colorir — deveremos examinar várias outras funções. O menu já nos deu uma idéia das diferentes opções, mas ainda não tivemos acesso a elas.

Carregue o programa anterior e adicione as linhas fornecidas neste artigo. Cada rotina é seguida de notas e sugestões de como tirar o melhor proveito das novas possibilidades do seu programa.

```
4000 REM retangulo e caixa
4010 LET caixa=0: GOTO 4030
4020 LET caixa=1
4030 FOR n=1 TO 50: NEXT n
4040 GOSUB 8000
4050 IF INKEY$=CHR$ 13 THEN  RA
ND USR 65380: GOTO 1000
4060 IF INKEY$<>CHR$ 32 THEN  G
OTO 4040
4070 FOR n=1 TO 50: NEXT n
4080 LET xx=0: LET yy=0: LET hx
=x: LET hy=y
4090 GOSUB 8000: FOR n=1 TO 2:
PLOT hx,hy
4100 DRAW  OVER 1;0,yy: DRAW  O
VER 1;xx,0: DRAW  OVER 1;0,-yy:
 DRAW  OVER 1;-xx,0: NEXT n
4110 IF INKEY$<>CHR$ 32 THEN  G
OTO 4090
4120 PLOT hx,hy: DRAW 0,yy: DRA
W xx,0: DRAW 0,-yy: DRAW -xx,0
4130 IF caixa=0 THEN  GOTO 4030
4135 IF xx=0 THEN  GOTO 4040
4140 FOR n=hx TO hx+xx STEP SGN
 xx
4150 PLOT n,hy: DRAW 0,yy: NEXT
 n
4160 GOTO 4040
5000 REM circulo
5010 FOR n=1 TO 50: NEXT n
5020 GOSUB 8000
5030 IF INKEY$=CHR$ 13 THEN  RA
```

```
ND USR 65380: GOTO 1000
5040 IF INKEY$<>CHR$ 32 THEN  G
OTO 5020
5050 FOR n=1 TO 50: NEXT n
5060 LET xx=0: LET yy=0: LET hx
=x: LET hy=y
5070 GOSUB 8000: CIRCLE  OVER 1
;hx,hy,ABS xx: CIRCLE  OVER 1;h
x,hy,ABS xx
5080 IF INKEY$<>CHR$ 32 THEN  G
OTO 5070
5090 CIRCLE hx,hy,ABS xx: GOTO
5000
5500 REM apagar
5510 GOSUB 8000
5520 IF POINT (x,y)=1 THEN  PLO
T  OVER 1;x,y
5530 IF INKEY$=CHR$ 13 THEN  RA
ND USR 65380: GOTO 1000
5540 GOTO 5510
6000 REM copiar
6010 COPY : GOTO 1000
7000 INPUT "INTRODUZA O NOME ";
 LINE n$: IF LEN n$>10 THEN  GO
TO 7000
7010 LOAD n$CODE 50000: RAND US
R 65368: GOTO 1000
7500 INPUT "INTRODUZA O NOME ";
 LINE n$: IF n$="" OR LEN n$>10
 THEN  GOTO 7500
7510 SAVE n$SCREEN$ : GOTO 1000
```

Selecione a opção RETANGULO e mova o cursor até onde deseja colocar um canto da figura que pretende desenhar; em seguida, pressione <SPACE>. Desloque o cursor para o canto diagonalmente oposto. Enquanto executa o movimento, um retângulo ficará piscando, para lhe dar a idéia exata do que está desenhando. Você pode, assim, escolher o tamanho e a forma ideal, simplesmente passeando com o cursor. Quando tudo estiver do seu agrado, pressione <SPACE>, para fixar o desenho.

A opção CAIXA funciona da mesma maneira que RETANGULO, só que a área interna é preenchida.

CIRCULO é a outra forma que você pode desenhar. Coloque o cursor no local que será o centro do círculo e pressione <SPACE>. Em seguida, mova-o para qualquer ponto da periferia e pressione novamente a tecla de espaço.

A opção COPIAR envia para sua impressora a imagem da tela. Depois de selecioná-la, responda às perguntas apresentadas. Terminado o trabalho, o programa retornará ao menu.

Para fazer correções e mudanças, use as opções APAGAR ou OOPS. Pequenos detalhes podem ser corrigidos com a primeira opção. Quando terminar, tecle <ENTER> para retornar ao menu.

Para mudanças radicais, utilize OOPS, que apagará tudo o que foi feito depois de sua última visita ao menu.

Os dois itens restantes são GRAVAR e CARREGAR (na fita, somente). Ao selecionar um dos dois, um nome será solicitado. Você pode CARREGAR sem especificar um nome de arquivo, simplesmente pressionando <ENTER>, mas, para GRAVAR, o nome é sempre necessário.

```
4000 SCREEN 1,ST:GOSUB 1500
4010 IF EF=1 GOSUB 500:RETURN
4020 IF PEEK(345)<>PC THEN 4000
4030 XS=X:YS=Y
4040 GOSUB 1500:GOSUB 500
4060 IF EF=1 THEN RETURN
4070 IF ABS((XS-X)*(YS-Y))>2300
0 THEN 4040
4080 LINE(X,Y)-(XS,YS),PSET,B
4090 IF PEEK(345)<>PC THEN 4040
4100 PMODE MD,5:GET(X,Y)-(XS,YS
),CP,G:PMODE MD,1
4110 XS=XS-X:YS=YS-Y
4120 GOSUB 1500:GOSUB 500
4130 IF EF=1 THEN RETURN
4140 IF(X+XS)<0 OR (X+XS)>255 O
R(Y+YS)<0 OR (Y+YS)>191 THEN 41
20
4150 LINE(X,Y)-(X+XS,Y+YS),PSET
,B
4160 IF PEEK(345)<>PC THEN 4120
4170 GOSUB 500:PUT(X,Y)-(X+XS,Y
+YS),CP,PSET:GOSUB 510
4180 GOTO 4120
5000 CLS:PRINT" SELECIONE A COR
 DA BORDA (0-8) "
5010 A$=INKEY$:IF A$<"0" OR A$>
"8" THEN 5010
5020 BC=VAL(A$):SCREEN 1,ST
5030 GOSUB 1500:GOSUB 500
5040 IF EF=1 THEN RETURN
5050 IF PEEK(345)<>PC THEN 5030
5060 PAINT(X,Y),CL,BC:GOSUB 510
:GOTO 5030
6000 CLS:PRINT" SALVAR OU CARRE
GAR DO GRAVADOR (S/C) ?"
6010 A$=INKEY$:IF A$<>"S" AND A
$<>"C" THEN 6010
6020 IF A$="S" THEN 6100
6030 PRINT" CONFIRMA QUE QUER C
ARREGAR       OUTRO DESENHO (S/N
)?"
6040 A$=INKEY$:IF A$<>"N" AND A
$<>"S" THEN 6040
6050 IF A$="N" THEN RETURN
6060 MOTORON:PRINT:PRINT " POSI
CIONE O TAPE,APERTE 'PLAY'  E E
NTAO PRESSIONE <ENTER>"
6070 IF INKEY$<>CHR$(13) THEN 6
070
6080 MOTOROFF:PRINT:PRINT:INPUT
"INTRODUZA O NOME DO ARQUIVO ";
A$
6090 SCREEN 1,ST:CLOADM A$:GOSU
B 510:RETURN
6100 PRINT:INPUT"NOME DO ARQUIV
O ";A$
6110 MOTORON:PRINT:PRINT" POSIC
IONE O TAPE,APERTE'RECORD' E EN
TAO PRESSIONE <ENTER>"
6120 IF INKEY$<>CHR$(13) THEN 6
120
6130 GOSUB 500:CSAVEM A$,1536,7
679,1536:RETURN
```

A opção RETANGULO permite que você desenhe retângulos de várias cores, em qualquer lugar da tela. Selecione a opção e mova o cursor até o ponto da tela onde você quer um canto da figura. Pressione a barra de espaço para identificar o ponto. Mova o cursor para o canto oposto. Da mesma maneira que com DESENHO e LINHA, você pode escolher cores ou abandonar a opção pressionando <ENTER>. Quando estiver satisfeito com o desenho, pressione novamente a barra de espaço.

CAIXA funciona como a opção anterior, mas a figura é preenchida com a cor em uso.

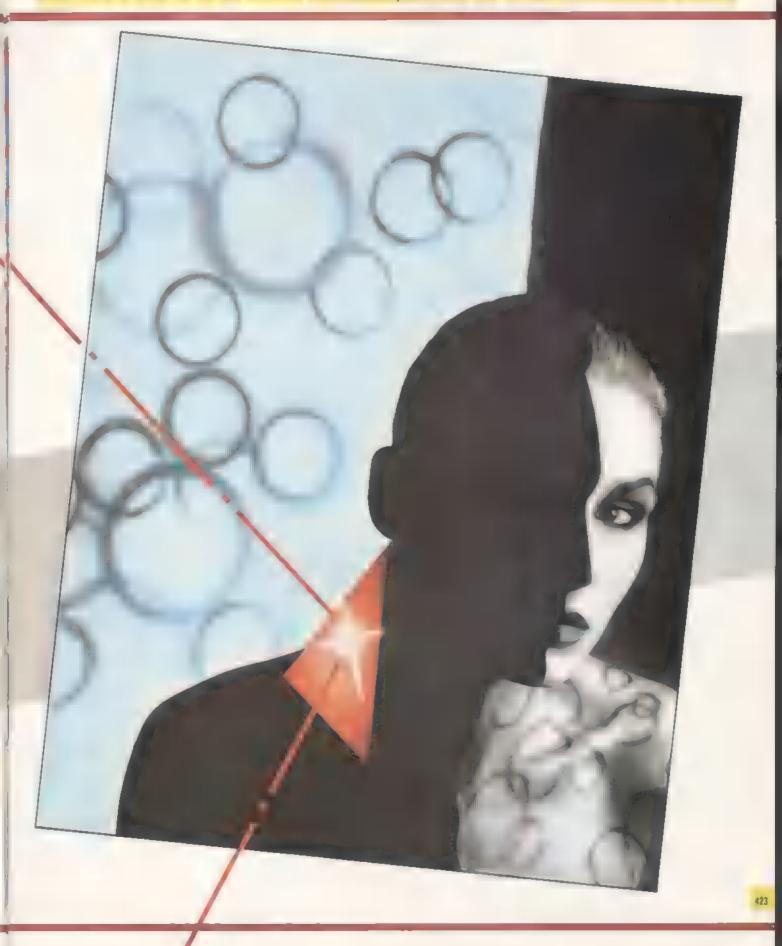
CIRCULO e DISCO são como RETANGULO e CAIXA. O primeiro ponto escolhido fica na periferia da forma. Mova o cursor, até chegar ao ponto desejado, que é o centro.

ELIPSE difere um pouco de CIRCULO. O primeiro ponto escolhido é o centro. Ao mover o cursor em qualquer direção, verá apenas uma linha. Mas, ao movê-lo na horizontal e depois na vertical, por exemplo, uma elipse começará a crescer. Ajuste-a até obter o que deseja e pressione a barra de espaço.

A opção COPIAR permite que você duplique uma imagem já desenhada na tela em outra parte dela. Para usá-la, posicione o cursor em um canto da área a ser copiada e pressione a barra de espaço. Ao mover o cursor em direção ao canto oposto, verá um retângulo que demarca a região onde o desenho será duplicado. Determine a área, quando estiver satisfeito com o seu tamanho e forma, pressionando novamente a barra de espaço. Mova essa área até o ponto desejado, usando as setas. Pressione a barra de espaço para efetuar a cópia, apagando o que havia naquela área. Até metade da tela pode ser copiada, sem problemas.

A opção PREENCHER funciona como o comando **PAINT** em BASIC. Ao selecioná-la, você deve informar a cor dos limites onde o **PAINT** deve parar. Mova, então, o cursor até a área a ser preenchida e pressione a barra de espaço. Escolha a cor pelas teclas 0 a 8.

Para corrigir possíveis erros, selecione a opção ERRO, que apaga tudo o que foi feito desde sua última visita ao menu. Pode-se, também, corrigir detalhes, selecionando a cor de fundo e apagando linhas ou pontos. Áreas maiores devem ser corrigidas com CAIXA ou DISCO.

A última opção permite que desenhos sejam carregados ou gravados na fita: conecte o cassete e responda às perguntas apresentadas.

```
3070  HPLOT PX(1),PY(1) TO PX(2
),PY(1) TO PX(2),PY(2) TO PX(1)
,PY(2) TO PX(1),PY(1):M = 0: GO
TO 3000
3080 RA =  SQR ((PX(1) - PX(2))
 * (PX(1) - PX(2)) + (PY(1) - P
Y(2)) * (PY(1) - PY(2))): FOR I
 = 0 TO 2 * PI STEP PI / 180
3090  HPLOT PX(2) + RA *  SIN (
I),PY(2) - RA *  COS (I)
3100  NEXT
3110  GOTO 3000
4000 CI = 0
4010  GOSUB 1500: GOSUB 1000: I
F A$ < > CHR$ (32) THEN  GOTO
 4010
4020  POKE M,ME
4030 CI = CI + 1:PX(CI) = X:PY(
CI) = Y: PRINT  CHR$ (7);: HPLO
T X,Y:M = 0
4040  IF CI < > 3 THEN 4010
4050 CE =  SQR ((PX(1) - PX(2))
 * (PX(1) - PX(2)) + (PY(1) - P
Y(2)) * (PY(1) - PY(2))):CE = C
E / 2
4060 A1 =  SQR ((PX(1) - PX(3))
 * (PX(1) - PX(3)) + (PY(1) - P
Y(3)) * (PY(1) - PY(3)))
4070 A2 =  SQR ((PX(3) - PX(2))
 * (PX(3) - PX(2)) + (PY(3) - P
Y(2)) * (PY(3) - PY(2))):AE = (
A1 + A2) / 2
4080 BE =  SQR (AE * AE - CE *
CE)
4090  FOR I = 0 TO 2 * PI STEP
PI / 180
4100  IF PX(1) < > PX(2) THEN
4120
4110  HPLOT PX(1) + BE *  COS (
I),PY(1) + ( - 1 ^ (PY(1) > PY(
2))) * CE - AE *  SIN (I): NEXT
 : GOTO 4000
4120  HPLOT PX(1) + ( - 1 ^ (PX
(1) > PX(2))) * CE + AE *  COS
(I),PY(1) - BE *  SIN (I): NEXT
 : GOTO 4000
7000  TEXT : HOME : PRINT "Grav
ar/Carregar"
7010  VTAB 10
7020  INPUT "Opcao? (G/C) ";R$
7030  IF R$ < > "G" AND R$ <
> "C" THEN 7030
7040  INPUT "Nome do desenho? "
;DE$
7050  INPUT "Pagina? ";P
7060  IF R$ = "C" THEN 7080
7070  PRINT  CHR$ (4);"BSAVE";D
E$;",A";P * 8192;",L";8192: GOT
O 250
7080  PRINT  CHR$ (4);"BLOAD";D
E$;",A";P * 8192: GOTO 250
```

Usaremos agora as opções mais sofisticadas do nosso editor de desenho. RETANGULO, CIRCULO e ELIPSE estão à sua disposição.

Para a opção RETANGULO, mova o cursor até um canto do quadrilátero que deseja traçar. Tecle a barra de espaço, leve o cursor para o canto diagonalmente oposto e tecle novamente a barra de espaço. Sua figura está pronta.

CIRCULO funciona de maneira parecida, só que o primeiro ponto corresponde ao centro da figura a ser traçada. O segundo determina o tamanho dela, ficando na periferia.

Ao usar a opção ELIPSE, não se esqueça de que o maior eixo da figura deve estar sempre na horizontal ou na vertical. Selecione três pontos para traçá-la. Os dois primeiros funcionarão como os focos da elipse e o terceiro ficará na periferia.

Por fim, GRAVAR/CARREGAR permite que você grave uma imagem em disquete ou que a carregue na memória do micro. É possível, inclusive, carregar um desenho pronto de algum outro programa ou jogo. A identificação de uma imagem de alta resolução no diretório é fácil: em geral, ela ocupa 34 setores do disco e tem a letra B à frente do nome do arquivo. Para carregá-la, simplesmente coloque o disquete no drive e digite o nome correto do arquivo quando este for solicitado pelo programa.

```
480 CI=0
490 GOSUB310:GOSUB210:IFA$<>CHR
$(32)THEN490
500 CI=CI+1:PX(CI)=X:PY(CI)=Y:B
EEP:PSET(X,Y),CO:M=0
510 IFCI<>2THEN490
520 ONOPGOTO590,590,530,540,550
,560,560
530 LINE(PX(1),PY(1))-(PX(2),PY
(2)),CO:M=0:GOTO480
540 LINE(PX(1),PY(1))-(PX(2),PY
(2)),CO,B:M=0:GOTO480
550 LINE(PX(1),PY(1))-(PX(2),PY
(2)),CO,BF:M=0:GOTO480
560 RA=SQR((PX(1)-PX(2))*(PX(1)
-PX(2))+(PY(1)-PY(2))*(PY(1)-PY
(2)))
570 CIRCLE (PX(1),PY(1)),RA,CO:
M=0:IFOP=6THENGOTO480
580 PAINT (PX(1)+1,PY(1)),CO:GO
TO480
590 LINE(PX(1),PY(1))-(PX(2),PY
(2)),CL,BF:M=0:GOTO480
600 CI=0
610 GOSUB310:GOSUB210:IFA$<>CHR
$(32)THEN610
620 CI=CI+1:PX(CI)=X:PY(CI)=Y:B
EEP:PSET(X,Y),CO:M=0
630 IFCI<>3THEN610
640 CE=SQR((PX(1)-PX(2))*(PX(1)
-PX(2))+(PY(1)-PY(2))*(PY(1)-PY
(2))):CE=CE/2
650 C1=SQR((PX(1)-PX(3))*(PX(1)
-PX(3))+(PY(1)-PY(3))*(PY(1)-PY
(3)))
660 C2=SQR((PX(3)-PX(2))*(PX(3)
-PX(2))+(PY(3)-PY(2))*(PY(3)-PY
(2))):AE=(C1+C2)/2
670 BE=SQR(AE*AE-CE*CE)
```

CORES NO SPECTRUM

O programa apresentado permite que você faça desenhos coloridos no Spectrum, mas é preciso ter cuidado para evitar problemas de sobreposição de cores. Se duas ou mais cores forem colocadas no mesmo retângulo de caractere, a última tomará o lugar das outras. Planeje seu desenho de maneira que cores adjacentes sejam alinhadas de acordo com esses retângulos. Lembre-se de que em cada linha temos 32 retângulos que contêm oito subdivisões na horizontal e 22 retângulos com oito subdivisões na vertical.

```
680 FORJ=0TO2*PI STEP PI/90
690 IFPX(1)<>PX(2)THEN720
700 IFPY(1)<PY(2)THENS=1ELSES=-
1
710 PSET (PX(1)+BE*COS(J),PY(1)
+S*CE-AE*SIN(J)),CO:NEXT:GOTO60
0
720 IFPX(1)<PX(2)THENS=1ELSES=-
1
730 PSET (PX(1)+S*CE+AE*COS(J),
PY(1)-BE*SIN(J)),CO:NEXT:GOTO60
0
```

Apagar permite que você suprima o desenho de determinadas áreas da tela. Selecione a opção e coloque o cursor no canto do retângulo que corresponde à área que pretende apagar. Pressione a barra de espaço. Leve o cursor para o canto diagonal oposto e pressione novamente a barra de espaço. Tudo o que estiver dentro do retângulo formado pelos 2 pontos será apagado. Tecle **<RETURN>** para trocar sua opção.

Retângulo e **Caixa** funcionam de maneira semelhante, sendo que a primeira opção desenha uma figura vazia, ao contrário da segunda. Ao selecionar uma delas, leve o cursor a um extremo da figura e tecle a barra de espaço. Desloque-o até a outra extremidade e repita a operação. Sua figura será desenhada instantaneamente.

Circulo e **Disco** têm uma relação semelhante à das duas opções anteriores. O primeiro ponto será o centro da figura e o segundo determinará o tamanho dela, ficando na periferia.

Elipse requer três pontos para cumprir sua função. Os dois primeiros determinam os focos da elipse e o terceiro, um ponto da periferia. Essa figura não é preenchida por cor. Para desenhos coloridos, você deverá recorrer à opção Pintar.

EDIÇÃO DE PROGRAMAS NO MSX

- COMO EDITAR LINHAS DE UM PROGRAMA BASIC
- O CONTROLE DO CURSOR
- TECLAS FUNDAMENTAIS E MOVIMENTOS MAIS COMPLEXOS

Os micros da linha MSX possuem sofisticados recursos de edição. Suas teclas de controle e seu cursor tornam possível ■ o vídeo como se fosse uma verdadeira "folha de papel".

Todos os interpretadores da linguagem BASIC oferecem alguns recursos para a edição de programas, ou seja, técnicas de entrada e alteração das linhas que os compõem. O primeiro interpretador BASIC, desenvolvido na década de 60 ■ Universidade de Dartmouth, nos EUA, fixou os recursos mais elementares de edição: numeração, inserção, apagamento ■ listagem de linhas. Estes foram posteriormente adotados em todos os outros dialetos do BASIC que surgiram.

O passo seguinte consistiu em dar ao programador a possibilidade de alterar caracteres, já que, com os recursos anteriores, o erro de um caractere obrigava-o a digitar novamente a linha toda.

Com esse objetivo, desenvolveram-se comandos como o **EDIT**, presentes nos micros da linha Sinclair, TRS-80 ■ TRS-Color. Embora os recursos de edição disponíveis no **EDIT** sejam bastante versáteis, eles ainda apresentam uma limitação: operam apenas em uma linha — cujo número precisa ser explicitado — de cada vez.

EDIÇÃO BIDIMENSIONAL

A limitação mencionada não passa de um resquício do tempo em que o BASIC era operado através de terminais impressores, desprovidos de movimentação bidimensional do cursor.

Os projetistas do MSX, porém, souberam superar essa limitação, abrindo a possibilidade de se editar um programa inteiro, e não uma linha de cada vez, desde que ■ trecho a ser editado estivesse exibido na tela. O comando **EDIT** foi, assim, suprimido.

O conceito de edição de programas no MSX aproxima-se, incidentalmente, do utilizado nos avançadíssimos micros profissionais da linha PC: envolve a utilização de teclas especiais, ou da combinação de determinadas teclas, para "passear" o cursor de texto pela tela, inserir e apagar caracteres em qualquer ponto da mesma, etc. O computador "lembra-se" das modificações realizadas na página de vídeo desde que se pressione a tecla **<ENTER>** ou **<RETURN>** após ■ término das modificações em uma determinada linha.

Uma possibilidade interessante do MSX é a de editar também os números de linha. Quando se utiliza esse recurso, obtém-se duas linhas: ■ que tinha o número original, em sua própria posição, ■ outra — que pode ser uma duplicata da anterior ou incluir modificações —, que é automaticamente inserida em sua nova posição.

As teclas fundamentais para a edição no MSX são as seguintes:

→	- desloca o cursor de texto em uma posição para ■ direita
←	- desloca o cursor de uma posição para ■ esquerda
↑	- desloca o cursor para cima
◁	- apaga o caractere imediatamente anterior ao cursor e recua o cursor de uma posição (retrocesso)
↓	- desloca o cursor para baixo
<INS>	- ativa ou desativa o modo de inserção de caracteres
<DEL>	- apaga um caractere e rejunta o restante da linha
<ENTER>	- consolida ■ modificações realizadas em uma linha

A tecla **HOME** também pode ser utilizada durante o processo de edição, para fazer ■ cursor retornar à posição no topo esquerdo da tela.

Ao editar um programa, ■ primeiro o comando **LIST**, para colocar ■ tela o trecho do programa que deseja editar. Em seguida, desloque o cursor até ■ linha a ser editada. Para fazer as modificações, escreva por cima do texto exibido, ou insira e apague caracteres por meio das teclas **INS** e **DEL**. Em seguida, pressione a tecla **<ENTER>** ou **<RETURN>**, se quiser preservar as modificações feitas.

A tecla **INS** em geral está desativada — ou seja, se você pressionar qualquer tecla de caractere, ele será impresso sobre o que estiver sob o cursor no momento. Para inserir um ou mais caracteres em determinado ponto de uma linha, primeiro desloque o cursor até lá. Em seguida, pressione uma vez a tecla **INS**, colocando o computador em modo de inserção. Para encerrar o processo, pressione **INS** novamente ou, então, **<ENTER>**.

Para verificar se o computador está em modo de inserção, observe o cursor de texto: de um retângulo, ele se transforma em um traço pequeno. Se você pressionar a tecla **DEL** durante o modo de inserção, poderá apagar caracteres sem desligar ■ inserção.

Com o auxílio da tecla **<CONTROL>**, pode-se obter movimentos mais complexos do cursor, assim como o acionamento de funções adicionais de edição. Pressionando-se simultaneamente **<CONTROL>** ■ uma outra tecla alfabética, chega-se a alguns resultados interessantes:

<CONTROL><F>	- desloca o cursor para a primeira letra da próxima palavra da linha
<CONTROL><B>	- desloca o cursor para a primeira letra da palavra anterior, na linha
<CONTROL><N>	- desloca o cursor para o final da linha
<CONTROL><E>	- apaga a linha corrente desde ■ ponto onde está o cursor até o final
<CONTROL><U>	- apaga toda a linha onde está o cursor

Convém lembrar, finalmente, que os recursos "normais" de edição de programas (numeração, inserção de linhas, renumeração, listagem, apagamento, etc.), presentes no editor padrão do BASIC, também funcionam no MSX.

PROGRAME UM CARTEADO

Microcomputadores podem [illegible] revelar grandes jogadores de baralho, com a vantagem de que nunca se cansam de jogar. Veja aqui como produzir gráficos para um carteado.

Seus amigos e parentes se negam a jogar baralho com você? Já se cansaram de perder dinheiro? O cacife [illegible] muito alto para seu orçamento? A solução para qualquer [illegible] desses problemas pode estar na série de artigos que iniciamos aqui. Programando seu micro para jogar Vinte-e-um, você terá uma vítima perfeita e poderá jogar sem perder um níquel.

Nesta primeira seção, veremos como programar os gráficos necessários para criar um baralho na memória — e na tela — de seu computador. O restante do programa — o jogo propriamente dito — será apresentado nos dois próximos artigos de *Programação de Jogos*. Grave o programa por partes, à medida que for sendo ampliado.

Se você não sabe jogar Vinte-e-um, não se preocupe: explicaremos as regras na última seção da [illegible]rie. Antes, porém, você precisará de um baralho completo.

A rotina gráfica que possibilita [illegible] computador mostrar [illegible] cartas [illegible] tela é a seguinte:

```
10 BORDER 4: PAPER 4: INK 9:
CLS : POKE 23658,8: LET B=0:
LET C=1
20 FOR N=USR "A" TO USR "R"+7
: READ A: POKE N,A: NEXT N
30 DIM C(52): FOR N=C TO 52:
LET C(N)=N: NEXT N
40 DIM A(13,13,2)
50 FOR N=C TO 10: FOR M=C TO
N: READ A(N,M,C),A(N,M,2):
NEXT M: NEXT N
60 FOR N=11 TO 13: LET A(N,C,
C)=4: LET A(N,C,2)=2: NEXT N
70 LET CC=C: LET CP=100
80 GOSUB 5000
500 LET Y=0: LET X=1
525 LET Z=C(CC)
530 GOSUB 5500
540 STOP
5000 CLS : PRINT AT 10,5;"EMBAR
ALHANDO AS CARTAS"
5010 FOR N=C TO 100
5020 LET X=INT (RND*52)+C
5030 LET Y=INT (RND*52)+C
5040 LET Z=C(X): LET C(X)=C(Y):
LET C(Y)=Z
```

- GRÁFICOS DE ALTA RESOLUÇÃO
- COMO DESENHAR AS CARTAS DO BARALHO
- POSICIONAMENTO DOS NAIPES
- COMO USAR VPOKE NA TELA GRÁFICA DO MSX
- COMO EMBARALHAR E DISTRIBUIR AS CARTAS

```
5050 NEXT N
5060 CLS : RETURN
5500 FOR N=Y TO Y+8: PRINT  PAP
ER 7;AT N,X:"     ": NEXT N
5510 LET ST=INT ((Z-C)/13)
5520 LET CH=144+ST
5530 LET VA=Z-(13*ST)
5540 IF ST<2 THEN  INK 2
5560 LET AC=147+VA
5600 PRINT  PAPER 7;AT Y,X;CHR$
 AC;AT Y,X+4;CHR$ AC;AT Y+8,X;C
HR$ AC;AT Y+8,X+4;CHR$ AC
5610 FOR N=C TO VA: IF A(VA,N,C
)<>B THEN  PRINT  PAPER 7;AT Y+
A(VA,N,C),X+A(VA,N,2);CHR$ CH
5620 NEXT N
5890 INK 9
5900 RETURN
9000 DATA 0,54,127,127,127,62,2
8,8,0,8,28,62,127,62,28,8,8,28,
62,127,127,62,8,28,8,28,28,107,
127,107,8,28
9010 DATA 0,8,20,34,34,62,34,34
,0,28,34,2,4,24,32,62
9020 DATA 0,28,34,2,12,2,34,28,
0,4,12,20,36,62,4,14
9030 DATA 0,62,32,32,60,2,34,28
,0,28,34,32,60,34,34,28
9040 DATA 0,62,34,2,4,8,16,16,0
,28,34,34,28,34,34,28
9050 DATA 0,28,34,34,30,2,34,28
,0,76,82,82,82,82,82,76
9060 DATA 0,14,4,4,4,4,36,24,0,
28,34,34,34,58,102,30,0,118,36,
40,48,40,36,118
9070 DATA 85,85,85,85,85,85,85,
85
9100 DATA 4,2
9110 DATA 2,2,6,2
9120 DATA 2,2,4,2,6,2
9130 DATA 1,1,1,3,7,1,7,3
9140 DATA 1,1,1,3,4,2,7,1,7,3
9150 DATA 1,1,1,3,4,1,4,3,7,1,7
,3
9160 DATA 1,1,1,3,2,2,4,1,4,3,7
,1,7,3
9170 DATA 1,1,1,3,2,2,4,1,4,3,6
,2,7,1,7,3
9180 DATA 1,1,1,3,3,1,3,3,4,2,5
,1,5,3,7,1,7,3
9190 DATA 1,1,1,3,2,2,3,1,3,3,5
,1,5,3,6,2,7,1,7,3
```

A linha 10 seleciona as cores da borda, dos caracteres e do fundo; **POKE** faz com que todas as letras sejam maiúsculas. As duas variáveis — **B** e **C** — são usadas no lugar de "0" e "1" no restante do programa. Devido à maneira como o Spectrum trabalha com números e variáveis, isso permitirá uma economia de 6 bytes, sempre que esses valores aparecerem. Assim, o programa poderá rodar no Spectrum de 16K.

A linha 20 prepara os caracteres (UDGs) usados nos naipes, números e letras de cada carta. Os dados para isso estão nas linhas **DATA** 9000 e 9070. Em seguida, a linha 30 cria um conjunto de 52 cartas não embaralhadas. A matriz **A**, dimensionada na linha 40, é preenchida com os valores das linhas **DATA** 9100 e 9190, que contêm as coordenadas das figuras dos naipes ■ cada carta. A linha 50 preenche parte da matriz com as coordenadas dos símbolos das cartas numéricas, enquanto a linha 60 preenche o restante com as coordenadas dos mesmos nas cartas com figuras. É possível criar desenhos para as cartas com figuras, mas seria bem cansativo e demorado digitar os dados necessários. Além disso, o programa ficaria grande demais para o Spectrum de 16K.

A linha 70 coloca os valores 1 e 100 nas variáveis **CC** ■ **CP**, respectivamente. **CC** é a carta atual, ou o elemento da matriz de cartas que o computador selecionou por último. **CP** é a quantidade de fichas que o jogador possui.

A linha 80 chama a sub-rotina que embaralha as cartas. Ela começa na linha 5000, que apaga a tela e comunica que as cartas estão sendo embaralhadas. O computador embaralha as cartas escolhendo duas, ao acaso, e trocando suas posições. O laço **FOR...NEXT** entre as linhas 5010 e 5050 repete o processo 100 vezes. Existe uma chance de que uma mesma carta seja escolhida "trocando de posição consigo mesma". Isso, na prática, não tem nenhuma importância, pois o número de trocas de posição é suficiente para garantir uma boa "embaralhada".

O valor da linha 5010 pode ser alterado para aumentar ■ número de trocas, mas números um pouco maiores que 100 já provocam uma demora inaceitável. A sub-rotina termina na linha 5060, que limpa a tela e retorna.

A linha 525 coloca na variável **Z** ■ valor da carta atual — elemento **CC** da matriz **C**. Em seguida, a linha 530 chama ■ sub-rotina 5500, que coloca as cartas na tela. A linha 5500 apresenta a parte branca da carta. A linha 5510 selecio-

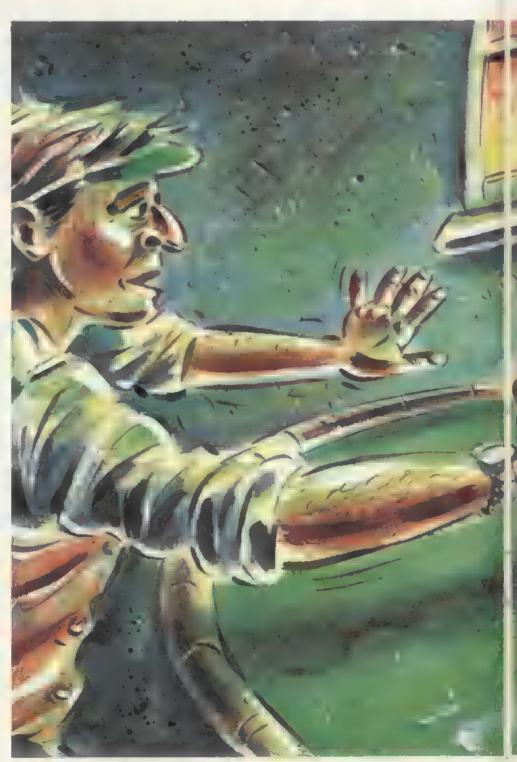
na o naipe correto (os naipes são numerados de 0 ■ 3). A linha 5520 calcula o código do caractere que contém ■ símbolo do naipe escolhido. A linha 5530 identifica a carta — afinal, o número de vezes que ■ símbolo do naipe vai aparecer depende disso.

Antes de desenhar ■ naipe, o computador seleciona a cor adequada. A linha 5540 atribui a cor vermelha aos naipes com número 0 ou 1, ■ a preta aos outros dois.

A linha 5600 desenha ■ número da carta usando o caractere adequado, escolhido pela linha 5560. A linha 5610 desenha os símbolos do naipe, cujas coordenadas são obtidas na matriz A.

Finalmente, com ■ escolha da cor 9 (cor de contraste), termina ■ sub-rotina.

Sprites são mais adequados para mover figuras. Como as cartas ficarão paradas, convém utilizar outra técnica para desenhar os números, letras ■ naipes das cartas.

```
10 CLEAR 500:COLOR 15,12,12:SCR
EENO:KEY OFF
20 LOCATE 8,11:PRINT "EMBARALHA
NDO AS CARTAS"
30 DIM NU$(13),NA(32):FOR K=1 T
■ 13:READ NU$(K):NEXT
40 DATA BD2S4U5ER3FD2NL4D3,RDLD
R,RDNLDL,DRDU2,NRDRDL,D2RULUR,R
S8DGD,ND2RDNLDL,NDRDNLD,D2S10BR
S12NU2RU2L,S4BD5F2R2U6L3R4,S4NR
5D6R3NH2NFR2U6,DNDS8RS12NEF
50 FOR I=1 TO 32
60 NA(I)=VPEEK(BASE(2)+23+I)
70 NEXT I
82 DIM SQ(61)
84 R=RND(-TIME):MN=100
■ SCREEN 2:FOR I=0 TO 3
90 VPOKE BASE(10)+252+I,32
92 VPOKE BASE(10)+252+I+256,32
94 VPOKE BASE(10)+252+I+512,32
96 NEXT I
120 FOR I=1 TO 32
130 VPOKE BASE(12)+2015+I,NA(I)
131 VPOKE BASE(11)+2015+I,(ABS(
I<17)*5+1)*16+15
132 VPOKE BASE(12)+4063+I,NA(I)
133 VPOKE BASE(11)+4063+I,(ABS(
I<17)*5+1)*16+15
134 VPOKE BASE(12)+6111+I,NA(I)
135 VPOKE BASE(11)+6111+I,(ABS(
I<17)*5+1)*16+15
140 NEXT
160 FOR K=0 TO 51:SQ(K)=K:NEXT
180 GOSUB 1500:N=0
190 FOR CX=31 TO 200 STEP 48:FO
R CY=1 TO 140 STEP 104
200 GOSUB 1000:GOSUB 2000:FOR J
=1 TO 500:NEXT J,CY,CX
210 FOR J=1 TO 1000:NEXT:GOTO 1
90
1000 ST=INT(SQ(N)/13)+1:NM=SQ(N
)-13*ST+14:N=N+1:IF N>51 THEN ■
=0
1010 RETURN
1500 FOR X=52 TO 2 STEP -1
1510 Q=INT(RND(1)*X)+1
1520 T=SQ(X-1):SQ(X-1)=SQ(Q-1):
SQ(Q-1)=T
1530 NEXT
1540 FOR X=0 TO 9:SQ(X+52)=SQ(X
):NEXT
1550 RETURN
2000 LINE (CX-1,CY-1)-(CX+42,CY
+71),15,BF
2005 LINE (CX-2,CY-1)-(CX-2,CY+
71),12
2010 S$=NU$(NM):PX=CX+24:PY=CY+
24:GOSUB 2550:PY=CY+15:GOSUB 25
50:PY=CY+35:GOSUB 2550:FOR J=0
```

```
TO 4:PX=CX+7:PY=CY+8+J*8:GOSUB
2550:PX=CX+34:GOSUB 2550:NEXT
2020 IF ST>2 THEN COLOR1 ELSE C
OLOR6
2030 DRAW "S12BM"+STRS(CX+3)+".
"+STRS(CY+2)+SS
2040 DRAW "S12BM"+STRS(CX+35)+"
."+STRS(CY+62)+SS
2050 IF NM/2<>INT(NM/2) OR NM>1
[illegible] THEN PX=CX+24:PY=CY+24:GOSUB
2500
2060 IF NM=2 OR NM=3 [illegible] NM=10 O
[illegible] NM=8 THEN PX=CX+24:PY=CY+15:G
OSUB 2500:PY=PY+19:GOSUB 2500
2080 IF NM<4 [illegible] NM>10 THEN 2140
2090 IF (NM=10 OR NM=8) THEN NS
=INT((NM-1)/2) ELSE NS=INT(NM/2
)
2100 FOR J=0 TO NS-1
2110 PX=CX+7:PY=CY+8+J*38/(NS 1
):GOSUB 2500
2120 PX=CX+34:GOSUB 2500
2130 NEXT
2140 RETURN
```

```
2500 ON ST GOTO 2510,2520,2530,
2540
2510 VPOKE BASE(10)+INT(PY/8)*3
2+INT(PX/8)+32,252:RETURN
2520 VPOKE BASE(10)+INT(PY/8)*3
2+INT(PX/8)+32,253:RETURN
2530 VPOKE BASE(10)+INT(PY/8)*3
2+INT(PX/8)+32,254:RETURN
2540 VPOKE BASE(10)+INT(PY/8)*3
2+INT(PX/8)+32,255:RETURN
2550 VPOKE BASE(10)+INT(PY/8)*3
2+INT(PX/8)+32,165:RETURN
```

A linha 10 cuida do espaço *string* da memória (retire o comando **CLEAR** para ver o que acontece), das cores da tela e das teclas de função ■ parte inferior do vídeo. A linha 20 comunica que as cartas estão sendo embaralhadas.

As letras ■ números das cartas serão desenhados com **DRAW**. A linha 30 coloca ■ instruções para ■ desenho dentro da matriz **NU$**, depois que ■ leu ■ linha **DATA** 40. A linha 30 também dimensiona essa matriz.

Como não podemos usar sprites, colocaremos os símbolos dos naipes (os caracteres 3 a 6) diretamente na tela de alta resolução, com **VPOKE**. A linha 50 buscará os desenhos desses símbolos dentro da tabela de padrões da tela de 40 colunas. O endereço inicial da tabela de padrões da tela ■ começa em **BASE (2)**.

A linha 60 dimensiona a matriz **SQ**,

usada para embaralhar ■ cartas, e, ainda, seleciona a tela de alta resolução e inicializa o gerador de números randômicos.

O laço **FOR...NEXT** das linhas 120 ■ 140 transfere os padrões dos símbolos dos naipes para a tabela de padrões da tela de alta resolução. Além disso, coloca as cores desses símbolos ■ tabela de cores.

Como a tela de alta resolução é dividida em três porções, tudo deve ser feito três vezes. Assim, as linhas 130, 132 e 134 cuidam dos padrões, enquanto ■ linhas 131, 133 e 135 ■ encarregam das cores.

O laço **FOR...NEXT** das linhas 70 a 110 modifica a tabela de nomes da tela de alta resolução, impedindo que os símbolos sejam desenhados agora. Apague essas linhas para ver ■ que acontece.

A linha 160 coloca valores de 0 a 51 nos primeiros elementos de **SQ**. A linha 180 chama a sub-rotina que embaralha as cartas (linhas 1500 e 1510). O valor 0 é colocado em **N**; isso significa que o primeiro elemento de **SQ** está sendo colocado em questão.

A linha 190 inicia um laço **FOR... NEXT** que usa as variáveis **CX** e **CY**, cuja função é posicionar os símbolos dos naipes nas cartas.

A linha 200 chama duas sub-rotinas. A primeira, da linha 1000, fornece o naipe — **ST** — ■ ■ valor — **NM** — da carta em questão. A segunda, da linha 2000 à 2140, calcula as posições onde os símbolos dos naipes devem ser desenhados naquela carta.

Esta última sub-rotina parece complicada mas, com ■ ajuda de um baralho, pode-se entender melhor seu funcionamento. Vários padrões se repetem na disposição dos símbolos dos naipes — as cartas de menor valor geralmente usam um só deles, enquanto ■ de maior valor requerem três ou mais. A subrotina verifica quais desses padrões são necessários para colocar os símbolos de determinado naipe em número ■ posição adequados.

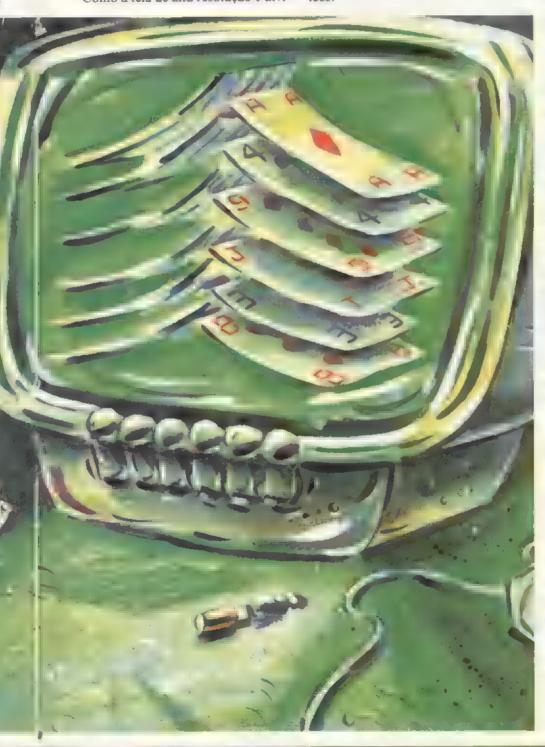
Antes do cálculo das posições dos naipes na tela, os números ou letras correspondentes ao valor da carta são desenhados em cantos opostos da mesma pelas linhas 2030 e 2040, que usam a informação contida em **NU$**.

As linhas 2050 a 2090, com base ■ valor da carta e de **CX** e **CY**, calculam as posições onde os símbolos dos naipes serão colocados na tela. As coordenadas da posição estão em **PX** e **PY**.

Uma vez que **PX** e **PY** tenham sido calculados, a sub-rotina da linha 2500 é chamada. Ela calcula ■ posição da tabela de nomes — **BASE (10)** — em que devemos colocar os símbolos dos naipes, para que eles apareçam numa posição correspondente a **PX,PY**. Essa subrotina é responsável pelo desenho propriamente dito.

A linha 2000 desenha a parte branca da carta, mas não apaga os desenhos produzidos com **VPOKE**. Isto é feito pela linha 2010, que apaga qualquer símbolo de naipe que tenha restado de uma antiga carta.

A linha 210 provoca uma pausa antes que o programa volte para a linha 190. Esta desenha uma nova série de oito cartas.

Apresentamos a seguir o programa que mostra as cartas na tela do Apple.

Para que funcione no TK-2000, precisaremos fazer uma pequena alteração.

```
10  HOME :E = 35000: HIMEM: E:
HTAB 9: VTAB 12: PRINT "EMBARAL
HANDO AS CARTAS"
20 F =  INT (E / 256): POKE 232
,E - F * 256: POKE 233,F
30  FOR I = E TO E + 41 + 17 *
32
40  READ A: POKE I,A
50  NEXT
60  SCALE= 1: ROT= 0
70  DATA  20 ,0 ,42 ,0 ,74 ,0 ,
106 ,0 ,138 ,0 ,170 ,0 ,202 ,0
,234 ,0 ,10 ,1 ,42 ,1 ,74 ,1 ,1
06 ,1 ,138 ,1 ,170 ,1 ,202 ,1 ,
234 ,1 ,10 ,2 ,42 ,2 ,74 ,2 ,10
6 ,2 ,138 ,2
72  DATA  [illegible] ,72 ,45 ,109 ,209
,251 ,219 ,23 ,77 ,73 ,169 ,251
,219 ,27 ,110 ,73 ,9 ,213 ,63
,63 ,63 ,55 ,77 ,73 ,169 ,251 ,
219 ,27 ,6 ,0 ,0 ,0
74  DATA  0 ,72 ,45 ,109 ,209
,27 ,223 ,155 ,73 ,9 ,77 ,218 ,
59 ,63 ,159 ,9 ,77 ,73 ,218 ,21
9 ,251 ,74 ,45 ,109 ,209 ,219 ,
219 ,19 ,0 ,0 ,0 ,0
76  DATA  0 ,72 ,45 ,109 ,209
,251 ,219 ,83 ,73 ,9 ,141 ,219
,63 ,255 ,74 ,73 ,105 ,218 ,223
,219 ,74 ,45 ,109 ,209 ,219 ,2
19 ,19 ,0 ,0 ,0 ,0 ,0
78  DATA  0 ,72 ,77 ,77 ,218
,251 ,251 ,74 ,77 ,77 ,218 ,63
,63 ,159 ,73 ,9 ,77 ,218 ,251 ,
219 ,74 ,73 ,77 ,218 ,219 ,219
,2 ,0 ,0 ,0 ,0 ,0
80  DATA  0 ,72 ,45 ,109 ,209
,219 ,27 ,159 ,9 ,77 ,73 ,218 ,
59 ,63 ,159 ,73 ,9 ,77 ,218 ,25
1 ,219 ,74 ,45 ,109 ,209 ,219 ,
219 ,19 ,0 ,0 ,0 ,0
82  DATA  0 ,72 ,45 ,109 ,209
,21[illegible] ,27 ,159 ,9 ,77 ,73 ,218 ,
59 ,63 ,159 ,9 ,77 ,77 ,218 ,25
1 ,251 ,74 ,45 ,109 ,209 ,219 ,
[illegible]9 ,19 ,0 ,0 ,0 ,0
84  DATA  0 ,72 ,45 ,45 ,141 ,
[illegible] ,223 ,219 ,74 ,73 ,105 ,218
[illegible] ,219 ,74 ,9 ,77 ,209 ,219
,223 [illegible]3 ,73 ,77 ,209 ,219 ,219
,19 [illegible] 0 ,0 ,0 ,0
86  [illegible] 0 ,72 ,45 ,109 ,209
,27 ,223 ,159 ,9 ,77 ,77 ,218 ,
59 ,63 ,159 ,9 ,77 ,77 ,218 ,25
1 ,251 ,74 ,45 ,109 ,209 ,219 ,
219 ,19 ,0 ,0 ,0 ,0
88  DATA  0 ,72 ,45 ,109 ,209
,27 ,223 ,159 ,9 ,77 ,77 ,218 ,
59 ,63 ,159 ,73 ,9 ,77 ,218 ,25
1 ,219 ,74 , [illegible] ,77 ,218 ,219 ,2
19 ,2 ,0 ,0 [illegible]
90  DATA  [illegible] ,45 ,173 ,2
51 ,251 ,187 [illegible] ,105 ,169 ,25
1 ,251 ,187 ,105 ,105 ,169 ,251
,251 ,187 ,105 ,41 ,45 ,213 ,2
19 ,219 ,19 ,0 ,0 ,0 ,0 ,0
92  DATA  0 ,72 ,9 ,45 ,141 [illegible]
19 ,223 ,155 ,73 ,9 ,77 ,218
51 ,219 ,74 ,73 ,77 ,218 ,251 ,
27 ,87 ,77 ,105 ,209 ,219 ,63 ,
159 ,0 ,0 ,0 ,0 ,0
94  DATA  0 ,8 ,45 ,45 ,109 ,2
18 ,223 ,27 ,87 ,77 ,9 ,141 ,27
,223 ,27 ,87 ,109 ,9 ,141 ,27
,223 ,31 ,87 ,45 ,45 ,109 ,218
,251 ,219 ,2 ,0 ,0
96  DATA  0 ,8 ,77 ,73 ,209 ,2
23 ,219 ,87 ,77 ,9 ,141 ,219 ,2
23 ,187 ,41 ,45 ,77 ,209 ,27 ,2
23 ,187 ,105 ,73 ,141 ,251 ,219

,187 ,0 ,0 ,0 ,0 ,0
98  DATA  0 ,72 ,109 ,109 ,26
,63 ,63 ,63 ,87 ,45 ,45 ,45 ,21
3 ,59 ,63 ,255 ,74 ,41 ,109 ,20
9 ,219 ,223 ,83 ,73 ,73 ,209 ,2
19 ,219 ,19 ,0 ,0 ,0
100  DATA  [illegible] ,72 ,9 ,77 ,209 ,
27 ,63 ,223 ,74 ,45 ,45 ,141 ,5
[illegible] ,63 ,63 ,191 ,9 ,45 ,45 ,141
,219 ,63 ,223 ,74 ,9 ,77 ,209 ,
219 ,219 ,19 ,0 ,0
102  DATA  0 ,72 ,41 ,109 ,209
,27 ,63 ,223 ,74 ,9 ,77 ,209 ,
255 ,31 ,191 ,41 ,45 ,45 ,173 ,
59 ,31 ,31 ,191 ,73 ,105 ,137 ,
219 ,63 ,223 ,2 ,0 ,0
104  DATA  0 ,72 ,9 ,77 ,209 ,
27 ,63 ,223 ,74 ,45 ,45 ,141 ,5
9 ,63 ,63 ,191 ,41 ,45 ,45 ,173
,27 ,31 ,31 ,159 ,73 ,105 ,137
,219 ,63 ,223 ,2 ,0
110  DIM SQ(61)
120  FOR K = 0 TO 51:SQ(K) = K:
NEXT
160  HGR : POKE  - 16302,0
170  HCOLOR= 1: FOR I = 0 TO 19
1: HPLOT 0,I TO 279,I: NEXT
180  GOSUB 1500:N = [illegible]
190  FOR CX = 6 TO 250 STEP 54:
FOR CY = [illegible] TO 120 STEP 100
200  GOSUB 1000: GOSUB 2000: FO
R J = 1 TO 500: NEXT J,CY,CX
210  FOR J = 1 TO 1000: NEXT :
GOTO 190
1000 ST =  INT (SQ(N) / 13) + 1
:NM = SQ(N) - 13 * ST + 14:N =
N + 1: IF N > 51 THEN N = 0
1010  RETURN
1500  FOR X = 52 TO 2 STEP  - 1
1510 Q =  INT ( [illegible] (1) [illegible] X) +
1
1520 T = SQ(X - 1):SQ(X - 1) =
SQ(Q - 1):SQ(Q - 1) = T
1530  NEXT
1540  FOR X = [illegible] TO 9:SQ(X + 52)
= SQ(X): NEXT
1550  RETURN
2000  HCOLOR= 3: FOR I = CY TO
CY + 80: HPLOT CX,I TO CX + 50,
I: NEXT
2010 S = NM
2020  HCOLOR= 0
2030  DRAW [illegible] AT CX + 3,CY + 8
2040  ROT= 32: DRAW S AT CX + 4
8,CY + 72: ROT= 0
[illegible]  IF NM / 2 <  > INT (NM /
2) O[illegible] NM > 10 THEN PX = CX + 2
0:P[illegible] = CY + 39: GOSUB 2500
2060  IF NM = 2 OR NM = 3 OR NM
[illegible] O[illegible] NM = 8 THEN PX = CX +
[illegible] = CY + 27: GOSUB 2500:PY
= PY + 26:PX = PX + 8: ROT= 32:
GOSUB 2500: ROT= 0
2080  IF NM < 4 OR NM > 10 THEN
2140
2090  IF (NM = 10 OR NM = 8) TH
EN NS =  INT ((NM - 1) / 2): GO
TO 2100
2095 NS =  INT (NM / 2)
2100  FOR J = 0 TO NS - 1
2110 PX = CX + 8:PY = CY + 20 +
J * 38 / (NS - 1):F = (J [illegible]
/ (NS - 1) + 20 > 39): ROT= F *
32:PX = PX + F * 8:PY = PY + F
* 2: GOSUB 2500: ROT= 0
2120 FE = (J * 38 / (NS - 1) +
20 < 34):F =  NOT (FE OR F):PX
= CX + 32 + ( NOT FE) * 8:PY =
PY + F * 2: ROT=  NOT FE * 32:
GOSUB 2500: ROT= 0
2130  NEXT
2140  RETURN
2500  ON ST GOTO 2510,2520,2530
,2540
2510  HCOLOR= 0: DRAW 14 AT PX,
PY: RETURN
2520  HCOLOR= 0: DRAW 15 AT PX,
PY: RETURN
2530  HCOLOR= 0: DRAW 16 AT  IN
T (PX), INT (PY): RETURN
2540  HCOLOR= 0: DRAW 17 AT  IN
T (PX), INT (PY): RETURN
```

Se você tiver um monitor monocromático e quiser dar um aspecto mais agradável à tela, apague a linha 170.

Os usuários do TK-2000 devem mudar a linha 160 para:

```
160 MP : HGR
```

As primeiras linhas do programa são responsáveis por montar na memória do Apple uma tabela de figuras contendo as letras e números das cartas, bem como os símbolos dos naipes. As linhas **DATA** 70 a 104 foram criadas pelo nosso programa editor de figuras. As primeiras delas, com exceção da mensagem "EMBARALHANDO AS CARTAS", já apareceram diversas vezes em *INPUT*.

A linha 110 dimensiona a matriz **SQ**, usada para embaralhar as cartas. A linha 120 enche essa mesma matriz com valores de 0 a 51.

A linha 160 liga a tela de alta resolução. No Apple, um **POKE** faz com que a tela seja totalmente utilizada. No TK-2000, [illegible] selecionada é a segunda, [illegible] **MP** precede **HGR**, para [illegible] mesmo efeito.

[illegible] desenha [illegible] de fundo [illegible] **jogo**.

A [illegible] na que embaralha [illegible] valor zero em **N**, [illegible] primeira carta em **SQ** [illegible] em questão.

[illegible] **FOR... NEXT** [illegible] as variáveis **CX** [illegible] para [illegible] cionar os símb[illegible] los [illegible].

A linha 200 chama duas sub-rotinas. A primeira, da linha 1000, fornece o naipe — **ST** — e ■ valor **NM** — da carta em questão. A segunda, da linha 2000 à 2140, calcula ■ posições em que devem ser desenhados os símbolos do naipe da mesma carta.

Esta última sub-rotina parece complicada mas, com ■ ajuda de um baralho, pode-se entender melhor seu funcionamento. Vários padrões se repetem na disposição dos símbolos dos naipes — as cartas de menor valor geralmente usam um só deles, enquanto as de maior valor requerem dois ou mais. A sub-rotina verifica quais desses padrões são necessários para desenhar os símbolos de determinado naipe em número e posição adequados. A instrução **ROT** complica um pouco mais essa sub-rotina, pois faz com que alguns símbolos apareçam de cabeça para baixo.

Antes do cálculo das posições, as linhas 2030 e 2040 desenham os números ou letras da carta em questão em cantos opostos da mesma, usando **DRAW**.

As linhas 2050 ■ 2090 calculam as posições de cada símbolo do naipe, com base no valor da carta. **PX** ■ **PY** são as coordenadas desta posição.

Uma vez que **PX** e **PY** tenham sido calculadas, a sub-rotina da linha 2500 é chamada. Essa rotina usa **DRAW** para desenhar os símbolos dos naipes.

A linha 210 provoca uma pausa antes que ■ programa volte para a linha 190. Esta desenha uma nova série de dez cartas.

T

Apresentamos ■ seguir a seção do programa para o TRS-Color que desenha as cartas. Digite-a e use **RUN** para ver o computador distribuir as cartas.

```
10 PMODE 3,1
20 CLS:PRINT @226,"ESTOU EMBARA
LHANDO AS CARTAS"
30 DIM NU$(13):FOR K=1 TO 13:RE
AD NU$(K):NEXT
40 DATA BD2S4U5ER5FD2NL5D3,RDLD
R,RDNLDL,DRDU2,NRD[illegible]L,RS8
DGD,ND2RDNLDL,NDR[illegible]RS1
2NU2RU2L,S4BD5F2[illegible]R5D
6R3NH2NFR[illegible] DND[illegible]
50 FOR [illegible] STE
P 32
60 RE[illegible]KE K,A PO[illegible] K+1,B
:NEXT
70 DA[illegible]6,
187,184[illegible]187,[illegible],
59,176,14,1[illegible]
80 DATA 2,0,3,[illegible]
[illegible]176,238,236[illegible],17[illegible],
[illegible]8,3,0,3,0
90 DATA 1,0,6,64,9,128,38,96,25
,144,102,100,153,152,102,100,15
3,152,34,32,1,0
100 DATA 2,0,9,128,6,64,9,128,3
4,32,153,152,102,101,153,152,10
2,101,17,16,2,0
110 DIM C(3),D(3),H(3),S(3),SQ(
61)
120 GET(0,0)-(13,10),H,G
130 GET(0,11)-(13,21),D,G
140 GET(0,22)-(13,32),S,G
150 GET(0,33)-(13,43),C,G
160 FOR K=0 TO 51:SQ(K)=K:NEXT
170 PCLS 6:SCREEN 1,1
180 GOSUB 1500:N=0
190 PCLS 6:FOR CX=6 TO 200 STEP
 50:FOR CY=11 TO 108 STEP 97
200 GOSUB 1000:GOSUB 2000:FOR J
=1 TO 500:NEXT J,CY,CX
210 FOR J=1 TO 1000:NEXT:GOTO 1
90
1000 ST=INT(SQ(N)/13)+1:NM=SQ(N
)-13*ST+14:N=N+1:IF N>51 THEN N
=0
1010 RETURN
1500 FOR X=52 TO 2 STEP -1
1510 Q=RND(X)
1520 T=SQ(X-1):SQ(X-1)=SQ(Q-1):
SQ(Q-1)=T
1530 NEXT
1540 FOR X=0 TO 9:SQ(X+52)=SQ(X
):NEXT
1550 RETURN
2000 LINE(CX,CY)-(CX+44,CY+72),
PRESET,BF
2010 S$=NU$(NM)
2020 IF ST>2 THEN COLOR 7 ELSE
COLOR [illegible]
2030 DRAW"S12BM"+STR$(CX+3)+","
+STR$(CY+2)+S$
2040 DRAW"S12BM"+STR$(CX+35)+",
"+STR$(CY+64)+S$
2050 IF(NM/2 <> INT(NM/2) AND N
M<>7) OR NM>10 THEN PX=CX+16: P
Y=CY+31:GOSUB 2500
2060 IF NM=2 OR NM=3 OR NM=10 O
R NM=8 THEN PX=CX+16: PY=CY+19:
GOSUB 2500: PY=PY+24: GOSUB 250
0
2070 IF NM=7 THEN PX=CX+16: PY=
CY+39: GOSUB 2500
2080 IF NM<4 OR NM>10 THEN 2140
2090 IF (NM=10 OR NM=8) THEN NS
=INT((NM-1)/2) ELSE NS=INT(NM/2
)
2100 FOR J=0 TO NS-1
2110 PX=CX+3:PY=CY+12+J*38/(NS-
1):GOSUB 2500
2120 PX=CX+30:GOSUB 2500
2130 NEXT
2140 RETURN
2500 ON ST GOTO 2510,2520,2530,
2540
2510 PUT(PX,PY)-(PX+13,PY+10),H
,OR:RETURN
2520 PUT(PX,PY)-(PX+13,PY+1[illegible]
,OR:RETURN
2530 PUT(PX,PY)-(PX+13,PY+1[illegible]),C
,OR:RETURN
2540 PUT(PX,PY)-(PX+13,P[illegible]0),[illegible]
,OR:RETURN
```

Escolhemos um **PMODE** com ■ cores, de modo que os naipes e números das cartas sejam azuis ou vermelhos. A linha 20 comunica ao jogador que as cartas estão sendo embaralhadas.

O comando **DRAW** coloca na tela de alta resolução as letras para os ases, reis, damas e valetes, bem como os números das cartas. As instruções para esse comando são lidas na linha **DATA** 40 e guardadas na matriz **NU$** pela linha 30, usando **READ**.

A linha 60 coloca os símbolos dos naipes na tela, usando **POKE**. Os padrões para esses símbolos são obtidos nas linhas **DATA** 70 ■ 90. As linhas 120 a 150 usam o comando **GET** para guardar os símbolos na memória, tão logo sejam desenhados. As matrizes que os guardam foram dimensionadas na linha 110, juntamente com ■ matriz **SQ**, empregada para embaralhar as cartas. A linha 160 coloca os números 0 ■ 51 na matriz **SQ**.

A linha 180 chama a sub-rotina que começa na linha 1500, para que as cartas sejam embaralhadas. O valor zero é colocado em N, o que significa que o elemento 0 da matriz **SQ** está em questão.

A linha 190 colore a tela de ciano e inicia dois laços **FOR...NEXT** que incluem as variáveis **CX** e **CY**, usadas posteriormente para posicionar os símbolos dos naipes nas cartas.

A linha 200 chama duas sub-rotinas. A primeira, da linha 1000, calcula o naipe — **ST** — e o número — **NM** — da carta em questão. A segunda, da linha 2000, calcula as posições em que os símbolos do naipe da carta serão desenhados.

Esta última sub-rotina parece complicada mas, com a ajuda de um baralho, pode-se entender melhor seu funcionamento. Existem certos padrões que se repetem ■ disposição dos símbolos [illegible] naipe — as cartas de menor valor usam apenas um deles, enquanto as de maior valor requerem dois ou mais. A [illegible] rotina verifica quais os padr[illegible]s necessários para desenhar a carta [illegible]estão.

Antes do cálculo das posi[illegible]o naipe, a letra ou o número da carta é desenhado em dois cantos opostos da mesma, com o comando **DRAW** e a informação contida em **NU$**.

As linhas 2050 ■ 2090 calculam ■ posições dos símbolos d[illegible] naipe com base no valor da carta. [illegible] **PY** são as coordenadas dessas [illegible]s, calculadas a partir de **CX** e [illegible]

Uma vez que [illegible]Y tenham sido calculadas, a sub-rotina 2500 é chamada, desenhando os símbolos do naipe.

A linha 210 provoca uma pausa an[illegible] que o programa volte à linha 190. Es[illegible]a apaga a tela e mostra outra carta.

FUNÇÕES MATEMÁTICAS

Aqui ■■■ mais algumas funções matemáticas em BASIC. Suas aplicações práticas vão ■■■ o cálculo ■■ área de um piso até o cálculo da velocidade de um corpo ■■ queda.

A função de potenciação tem variadas aplicações, sobretudo quando precisamos calcular áreas e volumes em nossos programas.

No computador, essa função, representada por ↑, é colocada sempre entre dois números. Por exemplo: 2↑3 (diz-se "dois elevado à terceira potência").

Na maioria dos computadores aqui indicados, ■ função de potenciação deve ser digitada por meio da tecla ^ (circunflexo), e é assim que ela aparecerá nas listagens. As exceções são:

ST

Usa-se ■ tecla ↑ para digitar a operação de potenciação.

Usa-se a tecla ** (duplo asterisco) para digitar a operação de potenciação.

Um número elevado à potência de outro é simplesmente ■ primeiro número multiplicado por ele mesmo um certo número de vezes. O segundo número determina quantas vezes.

De volta ao exemplo inicial, dois elevado à terceira potência é dois multiplicado por dois três vezes. Ou seja:

```
2^3 = 2*2*2 = 8
```

A operação é a mesma para qualquer par de números.

```
2^5 = 2*2*2*2*2 = 32
5^4 = 5*5*5*5 = 625
```

Convencionalmente, representa-se ■ função de potenciação colocando-se ■ segundo número, em tamanho menor, junto ■■ primeiro. Por exemplo, 2^5, 5^4. O computador, porém, não entende es■■ notação matemática.

AO QUADRADO E AO CUBO

A potência de dois ■ a potência de três recebem denominações especiais. Qualquer número elevado à potência de dois é dito "elevado ao quadrado", e qualquer número elevado à potência de três é dito "elevado ao cubo". De fato, existem razões para esses nomes.

Quando calculamos ■ área de um retângulo (seja ele ■■ tapete ou qualquer objeto retangular), medimos seu comprimento e largura e, em seguida, multiplicamos os dois números. O resultado dessa multiplicação é ■■ número em unidades quadradas. A área é medida em "metros quadrados" porque, na verdade, estamos calculando quantos quadrados, de lados iguais a um metro, cabem nesta área. Da mesma maneira, um número elevado a dois é dito ao quadrado (uma área quadrada é representada pela multiplicação da unidade básica por ela mesma).

Enquanto a potência de dois recebe um ■■■ de medida de área, ■ potência de três recebe um nome de medida de volume. Para calcular ■ volume de um cubo, multiplicamos o comprimento pela largura e, depois, pela altura. São necessárias, portanto, três multiplicações da unidade básica (duas para a área ■ mais uma para ■ altura). Um número elevado à terceira potência é, assim, um número ao cubo.

POTÊNCIAS MAIORES

Vimos que um número ao quadrado calcula área e ■■ número ■■ cubo calcula volume. Não é possível construir modelos para potências maiores que três. Se tivéssemos um objeto com quatro dimensões, seu volume seria medido em unidades de quarta potência, ■ que é obviamente absurdo. Contudo, ■■ potências maiores têm muita utilidade, como veremos a seguir.

Suponha, por exemplo, que queremos saber ■ probabilidade de um dado cair com a face seis para cima. É simples: cada face tem a mesma chance de aparecer; portanto, cada face tem 1/6 de chance. Suponha, agora, que queremos saber ■ probabilidade de obter seis duas vezes seguidas. Existe 1/6 de chance de que apareça um seis na primeira vez e, *se* aparecer, existe outro 1/6 de chance de que apareça um seis na segunda vez. A probabilidade total é 1/6 vezes 1/6, ou seja, 1/6 ao quadrado. O mesmo vale para quantas vezes quisermos. A chance de que o dado caia sete vezes com ■ três para cima, por exemplo, é: 1/6 * 1/6 * 1/6 * 1/6 * 1/6 * 1/6 * 1/6 = (1/6)↑7.

Quando examinamos a conversão para binário, tivemos a oportunidade de usar números elevados ■ potências maiores. Cada dígito binário representa uma potência do número dois. No número binário 11111111, por exemplo, o primeiro dígito da direita (bit 0) represen-

- A FUNÇÃO DE POTENCIAÇÃO
- NÚMEROS AO QUADRADO E NÚMEROS AO CUBO
- NÚMEROS ELEVADOS A GRANDES POTÊNCIAS
- O CÁLCULO DA ÁREA DE UM QUADRADO
- A RAIZ QUADRADA
- O USO DA FUNÇÃO SQR NUM PROGRAMA

ta um 1, o segundo (bit 1) um 2, o terceiro (bit 2) um 4 e os seguintes, respectivamente, 8, 16, 32, 64, 128.

Veja que 4 é 2*2, ou 2 elevado à potência de 2; 8 é 2*2*2, ou 2 elevado à terceira potência, 16 é 2*2*2*2 e assim por diante:

```
2^2 = 2*2             =  4
2^3 = 2*2*2           =  8
2^4 = 2*2*2*2         =  16
2^5 = 2*2*2*2*2       =  32
2^6 = 2*2*2*2*2*2     =  64
2^7 = 2*2*2*2*2*2*2   =  128
```

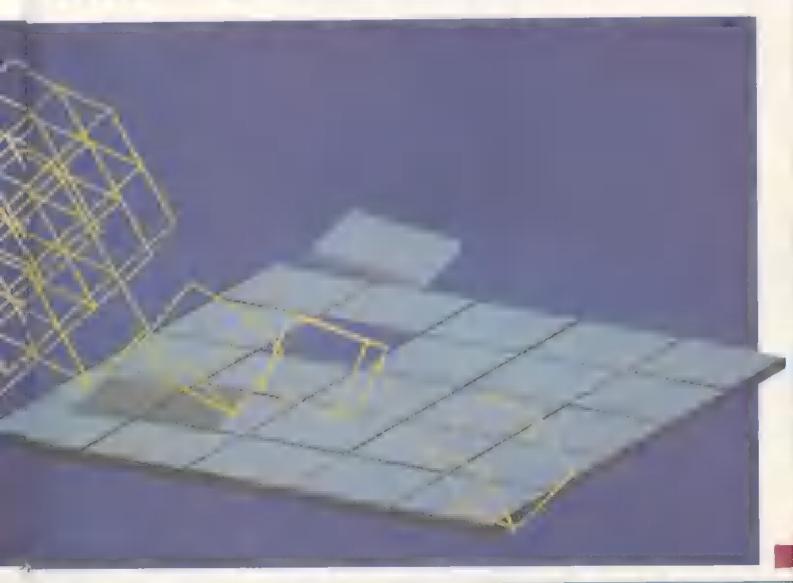
Estão faltando dois valores nessa tabela. Pode-se perceber facilmente quais são eles, mas a explicação talvez seja um tanto surpreendente. O primeiro valor é 2↑1. Obviamente, pela analogia binária, ele deve ser 2. Na verdade, se 2↑2 é 2 multiplicado por ele mesmo uma só vez, 2↑1 deve ser 2 multiplicado por ele mesmo uma vez ■ menos, ou seja, nenhuma vez. 2↑1 fica, então, valendo 2.

O valor abaixo deste é 2↑0. À primeira vista, parece que o resultado é zero mas, pela tabela acima, fica claro que é 1.

Isso acontece porque 2 precisa ser multiplicado por ele mesmo uma vez ■ menos que 2↑1. Como 2↑1 é 2 mesmo, só há uma maneira de multiplicar uma vez a menos: dividir 2 por ele mesmo. E qualquer número dividido por ele mesmo, como sabemos, vale 1.

Para verificar o que foi dito ou o valor de qualquer outra potência, você deve digitar:

```
PRINT 2^X
```

...onde X é valor que pretende verificar. Em seguida, tecle < **ENTER** > ou < **RETURN** >.

POR [illegible]

Experimente atribuir qualquer valor a X, ainda que valores grandes possam acarretar erros de sobrecarga (overflow). O que aconteceria, por exemplo, se se fizesse X igual a 1/2? Retornaremos ao assunto mais tarde. Por enquanto, vamos explorar um pouco mais a tão usada potência quadrada.

COMPARE QUADRADOS

O programa abaixo usa uma série de números para desenhar na tela diferentes quadrados, permite que se escolha dois deles e compara suas áreas. Acompanhando-o, a idéia de potência ficará mais clara.

```
10 SCREEN 2:COLOR 15,12,3
20 FOR N=17 TO 1 STEP -1
30 LINE(60,171)-(60+8*N,171-8*N
),1,B
40 NEXT
50 IF INKEY$="" THEN 50
60 CLS:SCREEN 0
70 PRINT"Deseja comparar alguma
 área (s/n) ?"
80 A$=INKEY$:IF A$<>"s" AND A$<
>"n" AND A$<>"S" AND A$<>"N" TH
EN 80
90 IF A$="n" OR A$="N" THEN 270
100 PRINT:INPUT"Qual [illegible] primeiro
 quadrado cuja área quer compar
[illegible] (1-17) ";A:A=INT(A):IF A<1 O
R A>17 THEN 100
110 PRINT:INPUT"E qual é o segu
ndo (1-17) ";B:B=INT(B):IF B<1
OR B>17 THEN 110
120 IF A>B THEN 220
130 CLS:PRINT" [illegible] primeira área
cabe ";
140 PRINTUSING"###.##";B^2/A^2;
150 PRINT" [illegible] den-tro da seg
unda"
160 FOR K=1 TO 2500:NEXT
170 CLS:SCREEN 2
180 LINE(60,171)-(60+8*A,171-8*
A),1,BF:LINE(60,171)-(60+8*B,17
1-8*B),1,B
190 IF INKEY$="" THEN 190
200 CLS:SCREEN 0:PRINT" Quer co
mparar mais alguma área (s/n) ?
"
```

```
210 GOTO 80
220 CLS:PRINT" A primeira área
é ";
230 PRINTUSING"###.##";A^2/B^2;
240 PRINT" vezes maior que a se
gunda"
250 SWAP A,B
260 GOTO 160
270 CLS
280 LOCATE 5,10
290 PRINT"qualquer tecla para r
einiciar"
300 IF INKEY$="" THEN 300
310 GOTO 10
```

```
10   HGR2 : HCOLOR= 3
20   HPLOT 10,180 TO 180,180
30   HPLOT 10,10 TO 10,180
40   FOR I = 10 TO 170 STEP 10
50   HPLOT 10,I TO 190 - I,I
60   HPLOT 190 - I,I TO 190 - I,
180
70   NEXT I
80   FOR T = 0 TO 3000: NEXT : T
EXT
```

```
90  HOME : VTAB 10
100  PRINT "DESEJA COMPARAR ALG
UMA AREA (S/N) ?"
110  GET A$: IF A$ <  > "S" AND
 A$ <  > "N" THEN 110
120  IF A$ = "N" THEN 500
130  PRINT : INPUT "QUAL O PRIM
EIRO QUADRADO QUE DESEJA COM-PA
RAR (1-17) ";A:A =  INT (A): IF
 A < 1 OR A > 17 THEN 130
140  PRINT : INPUT "E QUAL O SE
GUNDO (1-17) ";B:B =  INT (B):
IF B < 1 OR B > 17 THEN 140
150  IF A > B THEN 190
160  HOME : VTAB 10
170  PRINT "A PRIMEIRA AREA CAB
E ";B ^ 2 / A ^ 2;" VEZES DENTR
O DA SEGUNDA"
180  GOTO 210
190  HOME : VTAB 10
200  PRINT "A PRIMEIRA AREA E'
";A ^ 2 / B ^ 2;" VEZES MAIOR Q
UE A SEGUNDA"
210  FOR T = 0 TO 2000: NEXT
220  HGR2 : HCOLOR= 3
230 KA = (A + 1) * 10:LA = 190
- KA
240 KB = (B + 1) * 10:LB = 190
- KB
250  HPLOT 10,LA TO KA,LA TO KA
,180 TO 10,180 TO 10,LA
260  HPLOT 10,LB TO KB,LB TO KB
,180 TO 10,180 TO 10,LB
270  FOR T = 0 TO 3000: NEXT
500  TEXT : HOME : VTAB 10
510  PRINT "QUALQUER TECLA PARA
 REINICIAR";
520  GET A$: IF A$ = "" THEN 52
0
530  RUN
```

```
 10 CLS
 20 PRINT AT 2,0;"Comprimento"
 ;AT 3,0;"dos";AT 4,0;"lados"
 30 PRINT AT 2,24;"Area";AT 3,
 24;"do";AT 4,24;"quadrado"
 40 LET z=0: PLOT 55,23
 50 FOR n=1 TO 13
 60 LET m=n: IF m>7 THEN  LET
 m=m-8
 70 DRAW 0,n*8
 80 DRAW  PAPER m;n*8,0
 90 DRAW  PAPER m;0,-(n*8)
100 DRAW  PAPER 7;-(n*8),0
110 PRINT AT 6+(13-n),0;n;AT 6
+(13-n),25;n*n
120 PAUSE 10: NEXT n
140 INPUT "Voce quer comparar
areas (s/n)?";a$
150 IF a$<>"s" AND a$<>"n"
THEN  GOTO 140
160 IF a$="n" THEN  GOTO 300
170 INPUT "Qual e o primeiro q
uadrado cuja area voce quer co
mparar? (1-13)";a: IF a<1 OR a
>13 THEN  GOTO 170
180 INPUT "E qual e o segundo?
(1-13)";b: IF b<1 OR b>13 THEN
 GOTO 180
190 LET x=INT (a): LET y=INT (
b)
200 IF x>y THEN  GOTO 280
210 CLS : PRINT "A segunda are
a e ";y^2/x^2;" vezes maior do
 que a primeira"
220 PLOT 20,0: DRAW x*8,0:
DRAW 0,x*8: DRAW -x*8,0: DRAW
0,-x*8: DRAW y*8,0: DRAW 0,y*8
: DRAW -y*8,0: DRAW 0,-y*8
230 PRINT "Voce quer comparar
mais areas   (s/n)?"
240 INPUT a$
250 IF a$<>"s" AND a$<>"n"
THEN  GOTO 240
260 IF a$="n" THEN  LET z=1:
GOTO 300
270 GOTO 170
280 CLS : PRINT "A primeira ar
ea e ";x^2/y^2;" vezes maior d
o que a segunda"
290 GOTO 220
300 IF z=1 THEN  CLS : LET z=0
310 PRINT  INK 2; FLASH 1;AT 0
,0;" Pressione qualquer tecla
para   recomecar"
320 PAUSE 0
330 IF INKEY$="n" THEN  STOP
340 RUN
```

```
10 PMODE4,1:PCLS:SCREEN1,1
20 FOR N=17 TO 1 STEP -1
30 LINE (20,171)-(20+8*N,171-8*
N),PSET,B
40 NEXT
50 CLS
60 IF INKEY$="" THEN 60
70 PRINT" VOCE QUER COMPARAR AR
EAS (S/N) ?"
80 A$=INKEY$:IF A$<>"S" AND A$<
>"N" THEN 80
90 IF A$="N" THEN 220
100 PRINT:INPUT" QUAL E O PRIME
IRO QUADRADO CUJA AREA VOCE QUE
R COMPARAR (1-17)";A:A=INT(A):I
F A<1 OR A>17 THEN 100
110 PRINT:INPUT" E QUAL E O SEG
UNDO (1-17)";B:B=INT(B):IF B<1
OR B>17 THEN 110
120 IF A>B THEN 200
130 CLS:PRINT" A SEGUNDA AREA E
 ";B^2/A^2;"VEZES    MAIOR DO Q
UE A PRIMEIRA"
140 FOR K=1 TO 6000:NEXT
150 PCLS:SCREEN 1,1
160 LINE(20,171)-(20+8*A,171-8*
A),PSET,B:LINE(20,171)-(20+8*B,
171-8*B),PSET,B
170 IF INKEY$="" THEN 170
180 CLS:PRINT" VOCE QUER COMPAR
AR MAIS AREAS (S/N) ?"
190 GOTO 80
200 CLS:PRINT" A PRIMEIRA AREA
E";A^2/B^2:PRINT @32," VEZES MA
IOR DO QUE A SEGUNDA."
210 GOTO 140
220 CLS
230 PRINT @33,"PRESSIONE QUALQU
ER TECLA PARA   RECOMECAR"
240 IF INKEY$="" THEN 240
250 GOTO 10
```

A rotina que faz ■ comparação começa pedindo que se escolha o primeiro dos dois quadrados que serão cotejados. Em seguida, verifica se ■ número que entrou não é ilegal, ou melhor, se não é menor ou maior que o número de quadrados.

A função **INT** transforma em inteiro o número que entrou, caso este seja um decimal.

Uma vez escolhidos os dois números, o computador compara-os ■ fim de determinar qual é o maior. Em seguida, eleva cada número ao quadrado (isto é, multiplica cada um por si mesmo) e divide ■ maior pelo menor.

Finalmente, obtido o resultado, a rotina joga na tela a mensagem que diz qual número é o maior e ■ quanto ele é maior que o outro.

RAIZ QUADRADA

A potência de dois, ou a função quadrada, talvez seja a mais utilizada das potências. Mas muitas vezes precisamos usá-la no modo inverso, ou seja, achar o número que gerou ■ número ao quadrado que conhecemos. Suponha que sabemos, por exemplo, que a área de um quadrado é 81 e queremos especificar ■ comprimento dos lados.

Com ■ número 81 não é tão difícil: 9 vezes 9 é 81; portanto, o comprimento de cada lado deve ser 9. Mas ■ a área fosse 127, por exemplo, o cálculo do comprimento do lado (ou do número que ao quadrado é 127) torna-se bem mais difícil. Os computadores possuem uma função que ajuda neste cálculo: a função raiz quadrada.

Digite **PRINT SQR(81)** e tecle <**ENTER**> ou <**RETURN**> — o número 9 deverá aparecer na tela. Para saber quanto vale a raiz quadrada de 127, basta repetir a operação, substituindo o 81 dentro dos parênteses por 127.

O computador dispõe do comando especial **SQR** porque ■ raiz quadrada é amplamente empregada. Mas podemos utilizar também, para ■ mesmo cálculo, a função de potenciação.

Se você já usou frações como potência, deve ter observado que 2↑.5 (ou dois elevado a 1/2) tem o mesmo efeito que **SQR(2)**. Experimente em seu micro **SQR(81)** e depois 81↑.5. Os resultados podem não ser exatamente os mesmos mas, certamente, serão bem próximos.

A fração 1/3, usada como potência, calcula o inverso do cubo, ou seja, a raiz cúbica. A mesma analogia vale para as outras potências. Embora cada raiz tenha sua própria aplicação, a raiz quadrada é ■ mais usada de todas.

O comando **SQR** faz muito mais do que simplesmente calcular o lado de um quadrado do qual se conhece a área. Algumas equações matemáticas têm números ao quadrado e raízes quadradas e, nesses casos, pode-se utilizar **SQR** para calcular o resultado no computador.

O programa que apresentamos a seguir usa uma equação para calcular o tempo de queda de um objeto (desprezando-se a resistência do ar). Os físicos chamam essa situação de "corpo em queda livre no vácuo". O programa também calcula a velocidade em que ■ objeto ■ encontra quando atinge o solo. Esses cálculos envolvem números ao quadrado e raízes quadradas porque qualquer objeto em queda e sob a influência da gravidade cai cada vez mais rápido com o passar do tempo (desprezando-se ■ resistência do ar). Na verdade, estamos tratando com ■ quadrado do tempo. Mas, antes de maiores detalhes, vejamos o programa funcionando.

```
10 CLS
20 PRINT"Qual a altura da queda
 ";:INPUT D
30 IF D<0 THEN 20
40 T=SQR((2*D)/9.81)
50 V=SQR(2*D*9.81)
60 T=INT(T*100)/100
70 V=INT(V*100)/100
80 PRINT:PRINT"Tempo até chegar
":PRINT"ao solo =";T;"segundos"
90 PRINT:PRINT"Velocidade máxim
a":PRINT"de impacto =";V;"metro
s por segundo"
100 PRINT"(";1.6*INT(2.25*V+.5)
;" Km/h)"
200 LOCATE 4,20:PRINT"Qualquer
tecla para reinicio"
210 IF INKEY$="" THEN 120
220 RUN
```

```
10  HOME
20  PRINT "QUAL A ALTURA DA QUE
DA ";: INPUT D
30  IF D < 0 THEN 20
40 T =  SQR ((2 * D) / 9.81)
50 V =  SQR (2 * D * 9.81)
60 T =  INT (T * 100) / 100
70 V =  INT (V * 100) / 100
80  PRINT : PRINT "TEMPO ATE CH
EGAR": PRINT "AO SOLO = ";T;" S
EGUNDOS"
90  PRINT : PRINT "VELOCIDADE M
AXIMA": PRINT "DE IMPACTO ";V;"
 METROS POR SEGUNDO"
100  PRINT "(";1.6 *  INT (2.25
 * V + .5);" KM/H)"
200  VTAB 20: PRINT "QUALQUER T
ECLA PARA REINICIO"
210  GET A$: IF A$ = "" THEN 21
0
220  RUN
```

```
 10 CLS
 20 INPUT "INTRODUZA A DISTANC
 IA DA QUEDA (METROS)",D
 30 IF D<0 THEN  GOTO 20
 40 LET T=SQR ((2*D)/9.81)
 50 LET V=SQR (2*D*9.81)
 60 LET T=INT (T*100)/100
 70 LET V=INT (V*100)/100
 80 PRINT  INVERSE 1'''"TEMPO P
ARA CHEGAR AO CHAO:"; INVERSE
 0'T;" SEGUNDOS"
 90 PRINT  INVERSE 1'"VELOCIDA
DE MAXIMA ATINGIDA:"; INVERSE
 0'V;" METROS POR SEGUNDO"
100 PRINT "(";INT (2.25*V+.5);
" MPH)"
200 PRINT '''"PRESSIONE QUALQUE
R TECLA PARA   RECOMECAR"
210 IF INKEY$="" THEN  GOTO
 210
220 GOTO 10
```

```
10 CLS
20 INPUT "DIGITE A DISTANCIA DA
 QUEDA (EM METROS)";D
30 IF D<0 THEN 10
40 T=SQR((2*D)/9.81)
50 V=SQR(2*D*9.81)
60 T=INT(T*100)/100
70 V=INT(V*100)/100
80 PRINT @97,"tempo para chegar
 ao chao:":PRINT T;"SEGUNDOS"
90 PRINT @225,"velocidade maxim
a alcancada:":PRINT V;"METROS P
OR SEGUNDO"
100 PRINT " (";INT(2.25*V+.5);"
MPH)"
110 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA   RECOMECAR"
120 IF INKEY$="" THEN 120
130 GOTO 10
```

Em primeiro lugar, o computador limpa a tela e pergunta pela altura da qual o objeto vai cair. Caso você responda com um número negativo, a pergunta será feita novamente, pois não se pode calcular raiz quadrada de um número negativo.

Depois o computador calcula o tempo que o objeto levará para chegar ao chão e a velocidade com que chega. Isso é feito nas linhas 40 e 50.

A equação usada na linha 40 é uma das versões da "equação de movimento" que, depois de rearranjada, tomou a forma:

$$T = \sqrt{(2*D)/a}$$

...onde **T** é o tempo necessário para percorrer determinada distância (nesse caso, para atingir o solo), **D** é a distância (fornecida por você) e a é a aceleração (mede o quanto a velocidade aumenta em cada segundo).

No programa não há nenhuma variável a, já que seu valor é constante: no lugar de a vemos, assim, 9.81.

A linha 40 resolve a equação, indicando o tempo que o objeto leva para atingir o solo.

A linha seguinte resolve uma equação semelhante, que calcula a velocidade do objeto no momento em que atinge o solo.

$$V = \sqrt{2*D*9.81}$$

Nessa equação, em vez de dividir, o computador multiplica o número 2*D pela aceleração. V significa, no caso, velocidade.

Em ambas as equações, o sinal √ sobre o "2*D*9.81" ou "2*D39.81" significa raiz quadrada de — é aqui, portanto, que usamos a raiz quadrada. Uma vez obtida a resposta para uma das equações, pode-se mudá-la de modo que o computador calcule a altura da qual o objeto foi jogado.

De um modo geral, a equação se aplica a qualquer objeto em queda, seja ele um tijolo ou um foguete. O que pode mudar é o valor da aceleração. No nosso exemplo, a aceleração deve-se à gravidade; por isso, a é ajustado para este valor. A gravidade tem uma aceleração de 9.81 metros por segundo por segundo. Em outras palavras, para cada segundo na queda de um objeto sua velocidade aumenta de 9.81 metros por segundo. Metros por segundo por segundo também pode ser escrito como metros por segundo ao quadrado — e é aqui que a função potência entra em cena.

As equações para esse cálculo tomam a seguinte forma:

$$D = \frac{a*(t\hat{}2)}{2}$$

ou

$$D = V\hat{}2/(a*2)$$

Tente modificar o programa de modo que ele calcule respostas para estas equações. Em vez de altura, poderíamos fornecer ao computador a velocidade com que desejamos que um objeto atinja o solo (para a segunda equação) ou o tempo que ele deve levar antes do impacto. Em ambos os casos, o computador calcularia a altura necessária para satisfazer o dado fornecido.

A capacidade de resolução de problemas desse tipo tem vários usos práticos. Na maioria deles, porém, as equações necessitam de um tratamento mais cuidadoso, que leve em conta influências sobre o corpo em movimento. Alguns exemplos de aplicação seriam o cálculo do desempenho de um carro em testes de frenagem e aceleração ou mesmo a reconstituição de um acidente automobilístico. Neste último caso, poderíamos precisar a velocidade em que se encontrava o veículo no momento da batida. O cálculo é possível, contanto que conheçamos os valores corretos para satisfazer as equações.

Por meio da função de potenciação pode-se calcular muitas outras coisas além do desempenho de um carro ou da área de uma casa. Ela permite determinar, por exemplo, o quanto uma árvore cresce, ou por que um pássaro do tamanho de um elefante não pode levantar vôo. Num próximo artigo continuaremos a explorar o uso das funções matemáticas no computador, inclusive para a obtenção de efeitos gráficos.

Por que recebo uma mensagem de erro quando tento calcular a raiz quadrada de um número negativo?

Os computadores não são capazes de calcular a raiz quadrada de um número negativo e, na verdade, isto não existe no universo dos reais (embora para certos propósitos se atribua um valor imaginário).

Elevar um número ao quadrado significa multiplicá-lo por ele mesmo. Números positivos multiplicados entre si resultam num número positivo, mas o mesmo acontece com os negativos (negativo vezes negativo resulta num número positivo). Não existe nenhum número real que, elevado ao quadrado, resulte num número negativo. E é claro que o computador acusa erro quando lhe pedimos para fazer o impossível.

Quando usamos **SQR** num programa precisamos estar certos de que o número em questão será sempre positivo. Caso haja a possibilidade de que apareçam números negativos (resultados de cálculos anteriores, por exemplo), aconselha-se o uso da função **ABS**. Essa função converte um número em seu valor absoluto, ou seja, retira o sinal negativo, se houver. Para incluí-la num programa, substitua **SQR (A)** por **SQR(ABS(A))**, onde **A** representa o número que está usando.

COMO EVITAR ERROS

- USE MENSAGENS CLARAS
- ROTINAS DE PREVENÇÃO DE ERROS
- EXCLUA VALORES INCORRETOS
- INDICAÇÃO DE ERROS

Nem sempre é fácil separar o joio do trigo, quando ■ ■ ■ programação: programas inteiramente corretos podem apresentar erros inesperados. Aprenda ■ evitá-los.

A palavra-chave ■ uso. Não importa se ■ programa é considerado bom de um ponto de vista estritamente técnico: ■ ele ■ mostrar complicado, alguém vai fatalmente acabar tendo dificuldades e cometendo erros.

O segredo para contornar esse problema ■ fazer programas bem ordenados, de forma que todos os seus módulos sejam claros no que se refere à função ■ ■ modo de uso.

Isso significa oferecer muitas informações ■ telas explicativas e proteger adequadamente tanto o programa quanto o usuário de erros simples.

O ideal seria colocar no programa proteções contra todos os erros possíveis. Teclas pressionadas equivocadamente, entradas irregulares, valores absurdos — tudo, enfim, que possa atrapalhar a correta execução de um programa deve ser evitado. Para isso, é necessário que se incorporem vários tipos de rotinas de verificação de erros e validação de entradas ao programa. Muitos dos programas mostrados por *INPUT* até o momento fazem uso desses artifícios.

MENSAGENS

Mensagens precisas são muito importantes para ajudar o usuário a compreender de que maneira deve responder quando apresentado, por exemplo, ■ um menu de opções.

Suponhamos, por exemplo, um menu que mostre as seguintes opções, típicas de um programa que manipula um banco de dados:

1. Criar um novo registro.
2. Alterar um registro existente.
3. Carregar um arquivo.
4. Gravar um arquivo.

Se for usada uma mensagem do tipo "Selecione uma opção:" ou "Faça sua escolha:", ■ usuário ficará ■ dúvida sobre ■ que fazer. Deve dar entrada ■ número da opção ou digitar uma ou mais letras do nome da opção?

Certamente, uma mensagem mais adequada seria "Indique ■ escolha (1-4)" ou "Tecle 1-4 para fazer sua opção".

Esse tipo de indicação deve ■ feito

ENTRADA MUITO LONGA

também em perguntas que exijam uma resposta simples do gênero sim ou não. Assim, em vez de usar alguma coisa do tipo "Tenta novamente?" — onde "sim" pode ser indicado por uma tecla não específica, ou pelo acionamento do botão de disparo do joystick, ou teclando-se ■ —, especifique as possibilidades: "Pressione o botão de disparo para jogar novamente", ou "Tenta novamente? (S/N)". Estas são formas muito mais diretas ■ claras de proceder.

Mesmo ■ mensagem "Pressione qualquer tecla para continuar" pode causar confusão. Vale ■ pena prepará-la a fim de evitar que alguém queira pressionar algo como <**BREAK**> ou <**STOP**> ou ainda <**ESCAPE**>, correndo, assim, o risco de terminar o programa. Por exemplo, nenhum problema pode acontecer com esta mensagem: "Pressione a barra de espaços para continuar".

Em alguns tipos de computador, a tecla ■ ser pressionada pode ser evidenciada por meio do caractere invertido na mensagem.

Outra boa opção (que é também uma excelente técnica de programa), não importa a espécie de mensagem, é desativar qualquer tecla que possa atrapalhar ou interromper a execução do programa (isso pode ser feito com ■ emprego de rotinas que excluam todas as entradas impossíveis).

Para recapitular, ou mesmo para uma simples resposta "sim" ou "não", pode-se usar algo como:

```
90 LET A$=INKEY$
92 IF A$="" THEN  GOTO 90
95 IF A$<>"S" AND A$<>"N"
THEN  GOTO 90
```

```
90 A$=INKEY$
95 IF A$<>"S" AND A$<>"N" THEN
90
```

```
10 GET A$
20 IF A$<>"S" AND A$<>"N"  THEN
GOTO 10
```

Por outro lado, rotinas bem mais sofisticadas podem ser usadas para invalidar a entrada de grupos de valores.

Outra maneira de tornar as coisas mais fáceis para ■ usuário é restringir ■ quantidade de dados a serem digitados. Por exemplo: se for suficiente pressionar apenas uma tecla, então prepare o programa de acordo com essa possibilidade. E, para evitar qualquer confusão, utilize sempre ■ mesmo tipo de mensagem e resposta ao longo de um programa comandado por menus. Assim, se ■ seleção da opção do menu de abertura for feita pela teclagem de um número, tende a usar o mesmo sistema ■ todos os outros menus.

Da mesma forma, empregue sempre o mesmo código para cada menu ou lista de opções. Se ■ terceira opção for "GRAVAR" em um ponto e "SAIR DO PROGRAMA" em outro, alguém pode não ficar muito satisfeito com o seu programa.

Essa regra deve também ser aplicada quando ■ entrada de dados for necessária (particularmente, quando houver uma longa seqüência). Quando os dados forem restritos a um pequeno grupo e tiverem um padrão simples, pode-se usar uma mensagem múltipla. Um exemplo de dados desse tipo é um arquivo de nomes e endereços, no qual cada nome é seguido de quatro linhas de endereço e um número de telefone.

Por outro lado, é necessário tomar cuidado com coisas mais complexas, como um arquivo de cadastro de clientes, onde há pouca semelhança entre os dados, ■ ■ número de entradas muda de registro para registro. Alguns campos podem mesmo ficar vazios.

Uma maneira de limitar ■ número de erros, neste caso, é fazer uma divisão da entrada de dados, por grupos ou individualmente. Assim, pode-se ter apenas uma mensagem para nome e endereço e outra para cada detalhe específico que vier depois.

Monte as mensagens para que elas apareçam, uma de cada vez, ou em diferentes cores, ou (pelo menos) bem espaçadas umas das outras. Limpar a tela após cada série de entradas é muito

mais eficiente em termos estéticos do que deixar a tela ir rodando para cima (*scroll*) indefinidamente.

Em alguns tipos de programa uma entrada não usual pode chamar uma sub-rotina particular. Assim, em um arquivo de dados, pode ser requerida uma informação adicional para determinados tipos de entrada. Nesse momento, se o usuário, razoavelmente familiarizado com a seqüência normal na qual os dados são entrados, não notar a nova série de entrada de dados, pode cometer diversos tipos de erro.

Nesses casos, são necessárias algumas rotinas de detecção de erros; mas é importante também deixar claro para o usuário que os dados requeridos mudaram. Isso pode ser feito por meio de uma mensagem piscando ou um vídeo inverso, ou mesmo de algum tipo de aviso sonoro, como um bipe, se o seu computador puder emitir sons.

DETECÇÃO DE ERROS

A maneira mais fácil de evitar que o programa incorpore erros é dar ao usuário a possibilidade de aceitar ou rejeitar os dados já digitados. Isso pode ser feito com o auxílio de uma mensagem do tipo: "OS DADOS ESTÃO CORRETOS? (S/N)". Neste caso, se você pressionar a tecla N provocará o reinício da rotina de entrada de dados. Mas isso só será necessário se uma grande quantidade de dados for manipulada.

Se você pretende incorporar a um programa uma rotina de confirmação dos dados, junte as entradas em grupos; desse modo, o programa ficará mais fácil de operar, evitando a repetição da confirmação a cada entrada.

No entanto, apesar de rotinas desse tipo serem eficazes no combate aos erros, os programas têm que ser protegidos contra entradas inadequadas.

Por exemplo, como evitar que letras e números sejam colocados juntos onde somente se requer um desses tipos de informação (ou letra ou número)? Deve-se sempre antecipar possibilidades de erro como esta quando se está montando uma rotina.

LIMITES DE TAMANHO

Na maioria dos programas de controle de arquivos, o tamanho das entradas tem um limite máximo, seja em número de caracteres, seja em valor numérico. Um programa que imprima etiquetas, por exemplo, tem os seus dados restringidos pelo tamanho de cada etiqueta. Afinal, não tem sentido fazer a digitação de um endereço de 25 ou de trinta caracteres se a entrada dispuser de espaço para apenas vinte.

Podemos usar mensagens ou mostrar o limite da entrada com indicadores como caracteres invertidos ou qualquer outro efeito visual que deixe claro qual é o limite máximo aceitável.

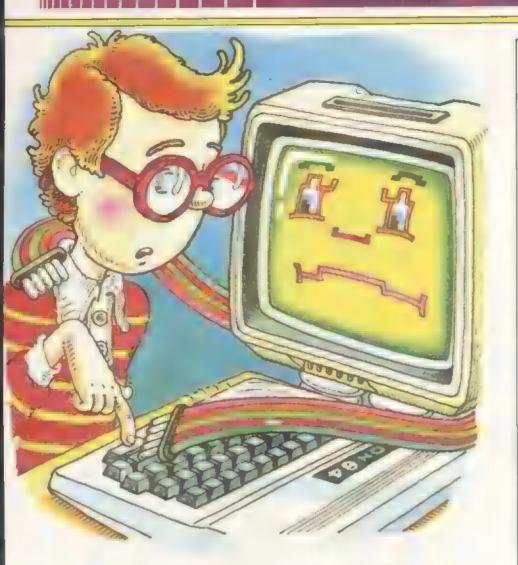
Ainda assim, rotinas adicionais devem ser usadas para invalidar entradas que excedam o máximo. Em alguns casos, pode-se truncar o dado no tamanho certo e deixar para a rotina de confirmação a tarefa de verificar se o dado será aceito ou não. Na maioria das vezes, porém, é melhor recomeçar a rotina de entrada no ponto onde houve erro, colocando uma mensagem como "ENTRADA MUITO LONGA! DIGITE NOVAMENTE." ou qualquer coisa parecida.

Outra boa razão para se limitar o tamanho das entradas é economizar memória. Afinal, não há sentido, por exemplo, em usar vinte ou mais caracteres para o código de endereçamento postal, quando este nunca ocupa mais de cinco. Procure diminuir o tamanho dos campos tanto quanto possível para economizar memória, especialmente no caso de programas de arquivo.

VALORES INCORRETOS

Estabelecer limites é importante também por outras razões, particularmente em programas que fazem uso de dados numéricos. Suponhamos, por exemplo, que o programa pede três números (ou os lê da memória), e divide a soma dos dois primeiros pelo terceiro. Um pequeno erro de digitação pode colocar um 0 no lugar de um 9 ou fazer com que o valor 0 apareça como resultado de outra operação. Nesse caso, a não ser que o programa esteja protegido contra esse tipo de erro, o resultado será uma mensagem "DIVISÃO POR ZERO" e o término abrupto do programa.

Num caso como o nosso, a proteção exigida é mínima, visto que uma simples rotina de verificação de entradas do teclado é suficiente para restringir entradas inoportunas.

Mas, no caso de cálculos "internos", onde é possível gerar um 0, devemos utilizar rotinas específicas. Este é um caso em que é preciso prever o pior. Assim, convém construir uma rotina com o uso de operadores relacionais (que serão abordados mais adiante, em outra lição). Podemos usar alguma coisa assim:

```
IF A=0 THEN GOTO 10000
```

A linha 10000 será então o início de uma rotina que poderá ajustar valor para algo diferente de 0 (mas aceitável pelo programa) ou avisar o usuário de que um cálculo impossível está para acontecer. Neste último caso, o programa será redirecionado para a rotina de entrada de dados para que se escolha um valor alternativo.

Uma gama específica de valores pode ser determinada usando-se simplesmente algo como o familiar:

```
IF N>100 OR N<1 THEN...
```

No caso de ser digitado um valor fora da faixa, instrução devolverá a execução para o início da rotina de entrada.

Qualquer que seja o seu procedimento dentro do programa, usuário deve ser avisado sobre qual é a faixa aceitável de dados e, se algum erro ocorrer, ele deve ser informado com uma mensagem adequada.

Desta maneira, ele saberá como proceder. Isso tudo pode ficar em sub-rotina acessada só em caso de necessidade.

INDICAÇÃO DE ERROS

Se você der ao usuário instruções claras de como proceder a cada vez que ele for confrontado com uma entrada de dados, uma grande parte dos erros será evitada. No entanto, quando algum erro ocorrer, deve-se informar exatamente a origem do problema para que ele não se repita.

Esse procedimento exige um conjunto de mensagens feitas especialmente para cada situação (não confundir com mensagens de erro do computador). Estas, por sua vez, podem ser definidas e colocadas em uma sub-rotina especial que só deve ser chamada quando acontecer algum problema.

Essas rotinas são normalmente usadas para indicar valores fora dos limites, duplicação de nomes de um campo-chave de um arquivo de dados, ou ainda entradas incorretas, seja pelo tamanho excessivo, seja pelo uso incorreto de caracteres. Assim, é possível criar mensagens para quaisquer tipos de erros em um programa.

Uma matriz e uma variável indicadora são suficientes para definir um certo número de mensagens fazer com que computador escolha a que adequada para a situação, após identificar o erro ocorrido.

A matriz para tais mensagens deve ser dimensionada logo no início do programa, como parte dos procedimentos de

MICRO DICAS

DETECTE ERROS AUTOMATICAMENTE

Uma das formas de evitar erros em um programa consiste em equipá-lo com dispositivos de detecção. Esses dispositivos funcionam no sentido de desviar o fluxo do programa, enviando-o a rotinas que corrigem a falha ou previnem o usuário de sua ocorrência.

Para isso, são necessárias instruções especiais. Os micros das linhas TRS-80, TRS-Color, Apple, TK-2000 e MSX, por exemplo, dispõem do pode- comando de desvio **ON ERROR GOSUB**... Este deve ser colocado em ponto do programa anterior bloco em que se deseja detectar erros execução. Normalmente, ele é digitado começo do programa. Sua função consiste em preparar o computador para "saltar" para linha indicada após o **GOSUB**, sempre que ocorrer um erro. O programador deve então escrever uma rotina (começando na linha indicada) que identifique o erro e adote medidas pertinentes. Para isso, existem as seguintes instruções suplementares:

- Função **ERR**: retorna o número de código do erro cometido.
- Função **ERL**: retorna o número da linha onde ocorreu a falha.
- Instrução **RESUME**: retorna a rotina de tratamento do para um ponto imediatamente após ao da ocorrência desse erro.

Os micros citados acima contam também com o comando **ERROR n**, que provoca uma indicação de erro num ponto do programa (n é o número do código). Esse comando testa a instrução **ON ERROR GOSUB** a rotina de tratamento de erros.

inicialização. Ela deve ser ajustada de tempos em tempos de modo a incluir novas mensagens prevenindo erros que possam vir a ocorrer. Desta maneira, se você resolver usar um conjunto de nove mensagens de erro, este poderia ser um exemplo:

```
10 DIM e$(9,20): FOR z=1 TO ■
: READ e$(z): NEXT z
20 DATA "Dado muito longo!","
Erro de digitacao!"
22 DATA "Senha errada!","Nao
confere!"
24 DATA "Introduza novamente!
","Nao toque!"
26 DATA "Pressione (s)im ou (
n)ao!","Somente numeros!","So
mente letras!"
```

```
10 DIM EM$(9):FOR Z=1 TO 9:READ
 EM$(Z):NEXT Z
20 DATA "DADO MUITO LONGO!","ER
RO DE DIGITACAO!"
22 DATA "SENHA ERRADA!","NAO CO
NFERE!"
24 DATA " REINTRODUZA OS DADOS!
","NAO TOQUE!"
26 DATA "PRESSIONE (S)IM OU (N)
AO!","SOMENTE NUMEROS!","SOMENT
E LETRAS!"
```

Obviamente, você escolherá mensagens adequadas às condições do seu programa. Assim, mais à frente no programa, a cada entrada de dados, pode ser feita ■ seguinte verificação:

```
1000 LET a$="": LET em=0: INPUT
 a$
1010 IF LEN a$>25 THEN LET em=
1

1010 IF a$<>"credito" THEN LET
 em=3
1010 IF a$="5" OR a$="9" THEN
LET em=4
1010 IF a$="" THEN LET em=5
1010 IF a$=" " THEN LET em=6
1010 IF a$<>"s" AND a$<>"n" THE
N LET em=7
1010 IF a$<"0" OR a$>"9" THEN
LET em=8
1010 IF a$<"a" ■ a$>"z" THEN
LET em=9
1010 FOR z=1 TO LEN a$: IF a$(z
)="0" THEN LET em=2
1015 NEXT z
```

```
1000 EM=0:INPUT A$
1010 IF LEN(A$)>25 THEN EM=1
1010 IF A$<>"CREDITO" THEN EM=3
1010 IF A$="5" OR A$="9" THEN E
M=4
1010 IF A$="" THEN EM=5
1010 IF A$=" " THEN EM=6
1010 IF A$<>"S" AND A$<>"N" THE
N EM=7
1010 IF A$<"0" OR A$>"9" THEN E
M=8
1010 IF A$<"A" OR A$>"Z" THEN E
M=9
1010 FOR Z=1 TO LEN(A$):IF MID$
(A$,Z,1)="0" THEN EM=2
1015 NEXT ■
```

Observe que ■ linha 1010 é opcional, ou seja, você pode escolhê-la ou não, dependendo da situação. Apenas a última opção utiliza uma linha adicional, a 1015. Atenção para a quinta opção, que funciona apenas no TRS-Color e no Spectrum.

Quando a checagem de entrada revela um erro, a variável **ME** assume um valor particular, relacionado com ■ mensagem previamente definida que corresponde ao erro.

A primeira opção testa ■ ocorreu uma entrada maior do que 25 caracteres; ■ segunda insiste na senha correta. A terceira é típica de uma reconfirmação de informação. A quarta pede que um dado seja redigitado após uma entrada vazia. A opção seguinte responde de modo inesperado quando a barra de espaços é pressionada. A sexta é uma variação da checagem que geralmente é feita após uma resposta sim/não. As duas seguintes verificam se apenas números ou letras foram digitados e ■ última testa a presença de um O (letra) ■ lugar de um 0 (número).

O programa prossegue, através de uma sub-rotina que mostra ■ mensagem adequada ao erro:

```
2000 IF EM>0 THEN PRINT TAB 6;
e$(em)
```

```
2000 IF EM>0 THEN PRINT EM$(EM)
```

COMPUTADORES QUE FALAM

■ existem sintetizadores de voz para a maioria dos microcomputadores domésticos. Veja aqui a descrição dos tipos mais importantes ■ os efeitos que você pode obter com cada um deles.

Nos filmes de ficção científica, vemos com freqüência computadores capazes de falar. Você deve se lembrar, entre vários outros, do HAL 9000, o tenebroso e megalomaníaco computador do filme "2001 — Uma Odisséia no Espaço", e os robôs falantes da série "Guerra nas Estrelas". Só muito recentemente, porém, os computadores reais e, particularmente, os microcomputadores pessoais receberam periféricos que lhes conferem o dom da voz. No exterior, já existem sintetizadores de voz para quase todos os tipos de microcomputadores — e a preços bastante baixos, graças à nova geração de chips de circuito integrado, fabricados especialmente para ■ síntese da voz. Alguns desses sintetizadores podem ser encontrados também no Brasil.

PARA QUE SERVEM?

Por que dar ao micro ■ capacidade de falar? Bem, antes de mais nada, não há dúvida de que isso seria tremendamente divertido. Basta pensar ■ infinidade de coisas que você faria com um computador falante: ele poderia contar piadinhas, recitar poemas ou, então, efetuar tarefas mais práticas, como ler em voz alta instruções complexas para um jogo, dispensando-o de abrir o manual ■ toda hora.

A utilização dos sintetizadores de voz em jogos é, também, muito interessante. Você já deve ter visto alguns videogames e flipers que tagarelam alegremente com ■ jogador. O sintetizador permite que você aumente a velocidade de interação com o jogo ■ que economize espaço na tela, se as mensagens ou instruções dirigidas ao jogador forem faladas, e não exibidas. Além disso, como você não precisará ler freqüentes mensagens ■ tela, poderá concentrar sua atenção nos joysticks e nos gráficos.

É claro que os sintetizadores de voz também têm aplicações mais "sérias", tais como auxiliar uma pessoa cega a utilizar o computador. Um programa especial pode fazer com que ■ sintetizador pronuncie, letra por letra, ou palavra por palavra, tudo o que é digitado no computador, ou que leia em voz alta listagens de programas ou dados, à medida que aparecem na tela.

Chips de síntese de voz estão começando ■ ser usados também em automóveis, para lembrar ■ motorista que ele esqueceu de colocar o cinto de segurança, avisar-lhe que ■ pressão do óleo está baixa, e assim por diante. E, ■ dúvida, podemos prever que ■ uso se expandirá grandemente no ramo dos eletrodomésticos. Já temos uma amostra disso nos fornos de microondas que avisam quando o prato está pronto, ou em secretárias eletrônicas que atendem educadamente aos telefonemas na ausência do morador.

Os sintetizadores de voz também são muito úteis ■ ensino, tornando divertido aprender a soletrar palavras em qualquer idioma, ■ ■ reconhecer letras ■ números.

Seu micro pessoal é capaz de tudo isso, desde que você conecte ■ ele o tipo apropriado de sintetizador de voz.

OS TIPOS PRINCIPAIS

Tecnicamente, existem dois métodos principais para se obter a síntese da voz humana. Eles são bem diferentes entre si, e ambos apresentam vantagens e desvantagens.

O primeiro método utiliza amostras de fala humana: um locutor enuncia letras, números, palavras ou frases inteiras, que são convertidas para números digitais e armazenadas permanentemente em um chip de memória. O segundo método armazena apenas sons elementares (fonemas, sílabas), que um gerador de sons do computador produz, utilizando-os depois para compor palavras e frases mais complexas.

O tipo de sintetizador que armazena amostras de fala humana alcança, naturalmente, uma qualidade de reprodução muito mais fiel. Em compensação, ocupa muito espaço na memória e seu vocabulário é mais restrito.

O chip sintetizador de voz desenvolvido pela Texas Instruments (já disponível no Brasil, com um vocabulário em português) é um bom exemplo deste primeiro tipo. As ondas sonoras (o sinal de voz) são inicialmente convertidas para números digitais por um conversor analógico-digital (ADC), de maneira que possam ser armazenadas em um chip de memória ROM. Posteriormente, quando a palavra ou frase é gerada, ■ sinal digital armazenado é convertido de volta para o sinal analógico original. Na verdade, não se armazena o sinal de voz original de forma integral, o que exigiria uma quantidade enorme de memória, mas apenas informações sobre o sinal. Consegue-se, assim, uma grande compactação do sinal original. Se a fala fosse integralmente digitada, precisaríamos de cerca de 500 kbytes para 2 minutos de fala — muito mais memória do que ■ maioria dos micros domésticos dispõe. Felizmente, a técnica de compressão obtém uma redução de 98%, de modo que ■ dois minutos cabem em um simples chip de ROM.

O segundo tipo de sintetizador de voz é o mais comum. Em vez de armazenar palavras completas, ele gera apenas os fonemas básicos do idioma, como "bá", "ô", "chi" etc. Como estes fonemas também são chamados de alofonos, o sintetizador tornou-se conhecido como sintetizador alofônico.

Existem cerca de sessenta fonemas deste tipo na maioria das línguas ocidentais. Com eles, pode-se construir qualquer palavra ou frase em uma determinada língua. É claro que os fonemas da língua portuguesa são bem diferentes dos da língua inglesa. Assim, em princípio, é possível sintetizar fala em português a partir de um sintetizador fabricado nos Estados Unidos, só que o computador terá um acentuado "sotaque" norte-americano.

Neste método, não se armazena a fala verdadeira, mas apenas a informação digital que é empregada para ativar o gerador de sons interno, o qual produzirá uma aproximação digital do fonema desejado. Cada fonema tem um código, ou endereço inicial de armazenamento, que está disponível em uma tabela de consulta dentro do chip.

Um programa escrito para o sintetizador deve preocupar-se apenas em chamar o código correto, quando ele é necessário. Assim, economiza-se muita memória, pois para cada segundo de fa-

COMO [illegible] O SINTETIZADOR

Quase todos os sintetizadores de voz são vendidos na forma de cartuchos (semelhantes à expansão de memória dos micros da linha Sinclair; são conectados ■ interface de expansão ■ parte de trás do console) ou na forma de placas de circuito impresso, que devem ser encaixadas em um dos soquetes internos do computador (é o caso dos micros da linha Apple). Os sintetizadores que ocupam o único soquete de expansão da máquina geralmente fornecem um soquete de expansão próprio, ao qual se pode ligar um joystick, uma impressora etc.

Outro tipo bastante comum de sintetizador é o que utiliza uma porta serial padrão, do tipo RS-232C, para se comunicar com ■ computador. Assim, podem ser usados com muitos tipos diferentes de microcomputadores.

Todos esses sintetizadores estarão prontos para operar tão logo sejam ligados ■ soquete correto. A saída do som pode ser feita através de uma conexão para um sistema de alta-fidelidade (amplificador e alto-falantes), ou então para a TV. Em ambos os casos, controla-se o volume e ■ altura do som gerado, o que possibilita torná-lo menos áspero ou "robótico". Outros tipos de sintetizador têm seu próprio sistema de amplificação e de alto-falante, ou utilizam o alto-falante interno do microcomputador. Nesse caso, geralmente é impossível controlar o volume sonoro.

SÍNTESE POR SOFTWARE

Os microcomputadores que possuem geradores sonoros sofisticados oferecem ■ alternativa bastante razoável para ■ obter síntese de voz sem necessidade de hardware especial. Uma máquina bem popular na Europa e nos Estados Unidos, o Commodore 64, tem um poderoso gerador de efeitos sonoros e musicais, que pode ser programado para sintetizar voz por aproximação digital. O sintetizador possibilita o controle de diversos canais de som, simultaneamente, bem como a manipulação de vários parâmetros de modulação de freqüência, envelope e intensidade.

No Brasil, os micros da linha TRS-Color e MSX podem ser usados desta maneira, ou seja, simplesmente adquirindo-se um software específico para síntese de voz. Evidentemente, o tipo utilizado, nesse caso, é o de síntese alofônica, ou por fonemas.

COMO PROGRAMAR O SINTETIZADOR

A forma mais direta de operação da maioria dos sintetizadores consiste, simplesmente, em pronunciar letras e números, cada vez que a tecla correspondente é pressionada no computador. Alguns modelos existentes para ■ Sinclair ■ o Spectrum também são capazes de pronunciar por extenso instruções, comandos e funções do BASIC, associados ■ cada tecla.

Entretanto, para fazer com que o sintetizador emita palavras ou frases completas, é necessário dispor de um software especial — geralmente em linguagem de máquina — que deve ser carregado na máquina antes da utilização do sintetizador. Este software habilita o interpretador BASIC, por meio de uma seqüência de comandos PRINT, a transmitir ao sintetizador ■ mensagem ou mensagens ■ serem pronunciadas.

Um sintetizador de voz para os micros da linha Apple, por exemplo, pode ser programado por meio de instruções **PRINT** comuns, dentro de um programa BASIC. Para isso, bastam alguns comandos **POKE** especiais, que habilitam a interface do sintetizador.

Existem três sistemas diferentes para especificar ■ seqüência de fonemas ou de vocábulos que devem ser pronunciados pelo sintetizador. O mais simples deles utiliza códigos numéricos para identificar os fonemas ou os vocábulos. Estes códigos dependem do chip sintetizador que está sendo empregado; a forma de usá-los dentro de um programa também varia amplamente. Por exemplo, os códigos para "hello" (alô) são 27, 7, 45 e 53. Os números podem ser entrados como uma linha **DATA**:

```
DATA 27,7,45,53
```

... ou como uma cadeia alfanumérica:

```
LET A$="27074553"
```

O manual que acompanha o sintetizador explica detalhadamente, com muitos exemplos, como utilizar os códigos.

A desvantagem desse sistema é que ele é pouco "natural", tornando muito trabalhosa ■ tarefa de especificar até mesmo seqüências curtas de fonemas ou de palavras. Por isso, desenvolveu-se ■ segundo tipo de software, mais fácil de usar, que se baseia em algo semelhante aos esquemas fonéticos de pronúncia encontrados nos dicionários. Por este método, as palavras que compõem a frase são escritas com notações especiais para cada fonema. Veja como você poderia dizer "hello" em dois tipos diferentes de sintetizador:

```
10 LET a$ = "he (ll)(oo)" :
PAUSE 1
```

```
10 A$="HH1.EH.LL.OW,4"
20 PRINT#A, A$
```

O manual do sintetizador traz uma lista de todos os fonemas, indicando como especificá-los. Como você vê, é bem mais fácil do que usar códigos numéricos mas, em compensação, é preciso conhecer detalhadamente a forma de especificação dos fonemas.

O terceiro sistema é bem mais sofisticado, fazendo a chamada "síntese por transformação direta texto-fala". Por enquanto, programas desse tipo existem somente para alguns idiomas, entre os quais ■ italiano, o alemão, o japonês, o francês e o inglês. Com eles, basta fornecer um texto escrito de forma correta ao sintetizador, e este produzirá automaticamente a seqüência de fonemas que gerará ■ voz:

```
10 CALL -435
20 PRINT "HELLO"
```

As regras de conversão texto-fala não são, naturalmente, infalíveis, e a existência de um grande número de exceções, notadamente no inglês, pode produzir os mais absurdos resultados.

Para usar em português um sistema como este, é necessário grafar as frases como um norte-americano as pronunciaria. Por exemplo, como será pronunciado o texto abaixo?

```
20 PRINT "EHDEEHTOHURAH NOHVAH
KOOLTOORAHL"
```

Alguns sistemas de síntese de voz permitem a programação de entonação ■ ênfase, de modo que ■ fala saia menos monótona e mais interessante, com pausas e modulações capazes de dar diferentes conotações às palavras — o que é muito importante na linguagem falada. Com isso, pode-se indicar ■ computador qual ■ sílaba tônica de uma palavra, se a frase é interrogativa ou exclamativa etc.

Como ■ vê, os sintetizadores, além de nos proporcionar momentos de diversão, poderão nos ensinar muita coisa sobre nossa própria linguagem, se forem do tipo alofônico.

O COMPUTADOR DÁ AS CARTAS

- A DISTRIBUIÇÃO DAS CARTAS
- COMO APOSTAR
- PEÇA MAIS CARTAS
- TOTAL DE PONTOS
- TENTE QUEBRAR A BANCA

Num luxuoso cassino, o homem da banca lança um olhar frio e indiferente. Você faz ■ aposta e recebe ■ nova carta. Há um momento de suspense. Será que você atingiu os 21 pontos?

Além da rotina gráfica apresentada no artigo anterior, precisamos de duas outras para jogar Vinte-e-um: uma que se comunique com o jogador e outra que permita ao computador fazer o papel da banca. A parte que você digitará agora cuida do jogador. Ela faz três coisas: dá as cartas, permite ao jogador fazer apostas e pedir mais cartas, e mostra ■ total de pontos — soma dos valores das cartas — e de fichas.

A banca solicita ao jogador que aposte, depois de ter dado duas cartas: uma para ele e outra para si mesma. Após receber a segunda carta, o jogador pode "pedir" outra, "comprar" mais uma ou "parar."

O programa tem rotinas para executar as tarefas correspondentes às opções do jogador, distribuindo novas cartas e/ou dobrando a aposta, ou passando ■ vez de jogar para a banca. Depois de dar uma carta, o programa verifica ■ o total de pontos ultrapassou 21, ou seja, ■ o jogador "estourou".

Além disso, o programa prevê duas outras situações: "natural" (quando o jogador faz 21 pontos com apenas duas cartas) e "mão de cinco" (quando cinco cartas reunidas não chegam ■ somar 22 pontos). Se qualquer delas ocorrer, o jogador será informado ■ a vez passará para a banca.

S

Adicione estas novas linhas ao programa do artigo anterior. Note que a linha 530 foi apenas aumentada.

```
 90 DIM S(2): LET BET=B
100 DIM O(2): DIM W(2)
500 LET Y=B: LET X=C: LET TF=B
: LET AF=B: LET PF=B: LET FF=B
520 FOR U=C TO 2
530 GOSUB 5500: GOSUB 6000:
 GOSUB 6500
535 IF CC=52 THEN  LET CC=B
540 LET O(U)=C(CC+C)
550 LET X=X+6
560 PRINT  PAPER 7;AT 21,B;"VO
CE TEM ";CP;" FICHAS "
570 IF U=C THEN  INPUT "QUAL E
A SUA APOSTA? ";AM: IF AM>CP
OR AM<C THEN  GOTO 570
590 IF U=C THEN  LET CP=CP-AM:
LET BET=AM
```

```
595 GOSUB 7000: GOSUB 7000:
NEXT U
622 IF S(2)=21 THEN  LET PF=C:
PRINT  PAPER 2;AT 4,18;" NATUR
AL ": GOTO 2500
700 IF (S(C)<16 AND (S(2)<16
OR S(2)>21) OR TF=C OR CP<AM)
THEN  GOTO 705
702 INPUT "COMPRA,PEDE OR SATI
SFEITO? "; LINE D$: IF D$<>"C"
AND D$<>"P" AND D$<>"S" THEN
GOTO 702
703 GOTO 715
705 IF (S(C)>21 OR CP<AM OR TF
=C) THEN  GOTO 710
706 INPUT "COMPRA OU PEDE? ";
LINE D$: IF D$<>"C" AND D$<>"P
" THEN  GOTO 706
708 GOTO 720
710 IF (S(C)<16 AND (S(2)<16
OR S(2)>21)) THEN  GOTO 770
711 INPUT "PEDE OU SATISFEITO?
"; LINE D$: IF D$<>"P" AND D$
<>"S" THEN  GOTO 711
715 IF D$="S" THEN  GOTO 2500
720 IF LEN D$=B THEN  GOTO 700
725 IF D$(C)<>"C" THEN  LET TF
=C: GOTO 905
760 LET BET=BET+AM: LET CP=CP-
AM
770 PRINT  PAPER 7;AT 21,B;"VO
CE TEM ";CP;" FICHAS"
905 LET Z=C(CC)
910 GOSUB 5500: GOSUB 6000:
GOSUB 7000
920 LET X=X+6
921 IF S(C)>21 THEN  GOTO 2000
925 IF X=31 AND S(C)<22 THEN
PRINT  PAPER 2;AT 21,B;" MAO D
■ CINCO! ": LET FF=C: GOTO
 2500
930 GOTO 700
2000 IF CP=B THEN  PRINT AT 21,
B;"VOCE PERDEU TODAS AS FICHAS!
": STOP
2010 IF CP>=1000 THEN  PRINT AT
 21,B;"PARABENS! VOCE QUEBROU A
 BANCA!": STOP
2020 PRINT #C;"PRESSIONE 'S' PA
RA OUTRA MAO"
2030 IF INKEY$<>"S" THEN  GOTO
2030
2037 IF PF=C ■■ DPF=C THEN  GOS
UB 5000
2040 CLS : GOTO 90
2500 STOP
6000 IF VA>10 THEN  LET VA=10
6010 LET S(C)=S(C)+VA: LET S(2)
=S(2)+VA
6020 IF VA=C AND AF=B THEN  LET
 S(2)=S(2)+10: LET AF=C
6030 IF S(C)>21 THEN  PRINT  FL
ASH C; PAPER 7;AT 20,B;" ";TAB
9;"VOCE QUEBROU! ";TAB 31;" ":
LET DPF=B: RETURN
6040 PRINT AT 20,B;" ";TAB 31;"
"
6050 PRINT  PAPER 7; INK B;AT 2
0,B;"SEU SCORE E' ";S(C);: IF S
(2)<>S(C) AND S(2)<22 THEN  PRI
NT  PAPER 7; INK B;" OU ";S(2)
6060 RETURN
6500 FOR N=10 TO 18
6510 PRINT  PAPER 7; INK 1;AT N
,X;CHR$ 161;CHR$ 161;CHR$ 161;C
HR$ 161;CHR$ 161
6520 NEXT N
6530 RETURN
7000 LET CC=CC+C: IF CC=53 THEN
 LET CC=C
7010 RETURN
```

As linhas 90 ■ 100 estabelecem ■ variável **BET** ■ três matrizes: **S**, que contém os dois totais do jogador (um ás vale 1 ou 11); **O**, que contém as duas primeiras cartas da banca; e **W**, que contém os dois totais correspondentes à soma dessas duas cartas.

A linha 500 estabelece as coordenadas do canto superior esquerdo da carta, e coloca o valor 0 em quatro variáveis indicadoras. **TF** indica se o jogador já pediu cartas. Quando isto acontece pela primeira vez, seu valor passa ■ ser 1 (segundo as regras, após "pedir" uma carta, o jogador não poderá mais "comprar" outras). **AF** é um sinalizador de ■■ usado para calcular os dois totais, já que um ás vale 1 ou 11. **PF** é um sinalizador de "naturais", ■ **FF** é um sinalizador de "mãos de cinco" (as regras do jogo serão apresentadas no próximo artigo).

O laço **FOR...NEXT** entre as linhas 520 e 595 cuida da distribuição das duas primeiras cartas do jogador e da banca. A linha 530 chama duas sub-rotinas: a da linha 6000 atualiza o total à medida que as cartas são dadas, e a da linha 6500 desenha ■ parte de trás das cartas da banca.

Analisemos essas sub-rotinas mais detalhadamente. A linha 6000 verifica se ■ carta em questão é uma carta de figura. Se for, seu valor será 10. A linha 6010 soma o valor da carta aos dois totais da matriz **S**. A 6020 verifica se a carta é um ás. A 6030 averigua e informa se o jogador "estourou".

Depois que o programa retorna ao seu curso principal, ■ linha 535 verifica se ■ carta atual ■ de número 52; se isso acontecer, o número da carta será mudado para 0 (nosso baralho tem 52 cartas, numeradas de 1 a 52). A linha 540 coloca as cartas da banca na matriz **0**. A 550 adiciona seis posições à coordenada X da carta seguinte.

O número de fichas que o jogador ainda tem é mostrado pela linha 560. A linha 570 pede para que seja feita a aposta e verifica se há fichas suficientes. O valor da aposta é colocado ■■ variável **BET** e subtraído do total de fichas pela linha 590. As variáveis **AM** e **BET** são necessárias para o caso de o jogador vir a "comprar" cartas.

Os dois **GOSUB** ■■■ da linha 595 selecionam a carta atual (CC), assegurando seu retorno ao início do monte de cartas para que este não acabe. O laço **FOR...NEXT** termina aqui.

A linha 622 verifica se as duas primeiras cartas do jogador fazem um "natural", ou seja, um total de 21 pontos. O sinalizador de "naturais", **PF**, se torna 1 e o programa pára (a linha 2500 será apresentada no próximo artigo).

As linhas 700, 705 e 710 testam as diversas combinações de totais de pontos, ■ indicador **TW** e o número restante de fichas. Sua função é selecionar as opções para o jogador. Por exemplo, suponhamos que o total de pontos do jogador seja maior que dezesseis e menor que 21. Se ele ainda não tiver pedido nenhuma carta e o número de fichas for

suficiente, a linha 720 colocará à [illegible] disposição as opções de "parar", "pedir" ou "comprar" cartas. Se, ao contrário, ele já tiver "pedido" alguma carta, não poderá mais "comprar" e apenas as opções de "parar" ou "pedir" mais cartas lhe serão dadas pela linha 711. A outra combinação de opções — "comprar" ou "pedir" mais cartas apenas quando nenhum dos totais em S ultrapassar dezesseis — é oferecida pela linha 706. Se o jogador resolver parar a [illegible] altura, essa versão inacabada do programa terminará.

Caso apenas <ENTER> tenha sido pressionado, sem a letra correspondente à opção, a linha 720 repetirá as opções. A linha 725 acionará o indicador **TW** caso o jogador não tenha "comprado" cartas. Se tiver, o programa irá para a linha 905.

Se o jogador preferir comprar uma carta, a linha 760 modificará o valor da aposta (**BET**) e do restante das fichas (**CP**). A linha 770 mostra quantas fichas sobraram. As linhas de 905 a 920 mostram as cartas restantes, que foram "compradas" ou "pedidas". A linha 921 atualiza os pontos do jogador. Caso ele tenha "estourado", o programa irá para a linha 2000, onde será feita [illegible] série de verificações.

A linha 925 verifica se o jogador tem uma "mão de cinco", examinando a posição da última carta dada ao jogador: se esta corresponder à quinta carta, e o total for inferior a 22, isso significa que o jogador tem uma "mão de cinco". O indicador **FF** será então acionado e o programa irá para a linha 2500. Se o jogador ainda não tiver parado de "pedir" cartas nem conseguido uma "mão de cinco", a linha 930 fará o programa voltar à linha 700.

QUEBRANDO A BANCA

Se o jogador perder todas as fichas, será informado pela linha 2000 e o jogo acabará. Se ganhar mais de 1000 fichas, ele "quebrará a banca", sendo informado pela linha 2010. Se isso acontecer, o jogo terminará.

Se o jogador tiver menos de 1000 fichas, a linha 2020 perguntará se ele deseja outra "mão". **PRINT 1** é usado para colocar a mensagem na primeira linha da parte inferior do vídeo.

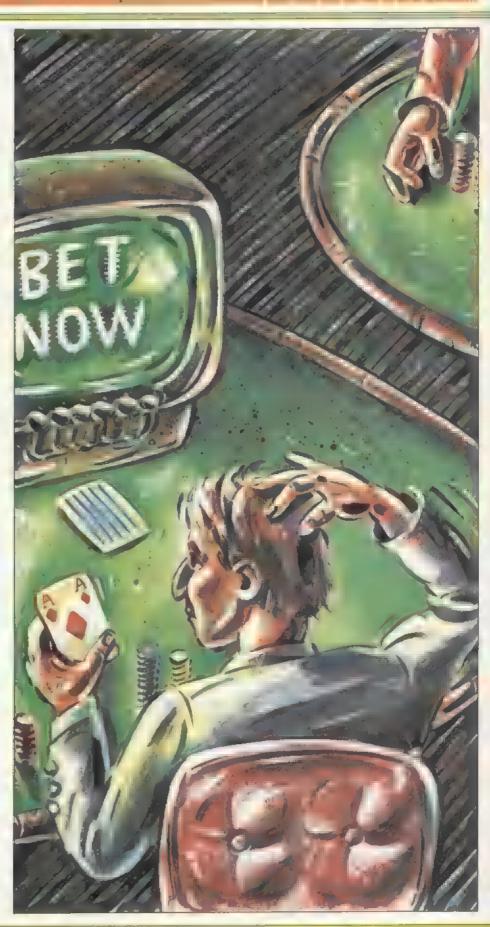
Quando um dos jogadores obtiver um "natural", as cartas serão embaralhadas novamente pela linha 2037, que chamará a sub-rotina adequada. Se ele quiser jogar outra vez a linha 2040 limpará a tela, reiniciando o jogo.

Apague as linhas 190 e 200 do programa anterior e adicione as próximas linhas.

```
84 R=RND(-TIME):MN=100
[illegible] CX=31:CY=1:PL=0:DL=0:NC=2:DA
=0:PA=0:P5=0:TW=0:PF=0:NA=N:BT=
0
210 GOSUB 3000
220 D1=N:GOSUB 1000:FOR I=107 T
O 180 STEP 3:LINE (30,I)-(74,I+
2),6-8*(I/2=INT(I/2)),BF:NEXT
240 GOSUB 5000:GOSUB 6500
250 GOSUB 8000:GOSUB 5000:GOSUB
 7000
260 GOSUB 5000:PRESET(8,80):PRI
NT#1,"Use as setas para apostar
.":PRESET(8,92):PRINT#1,"Depois
 aperte <RETURN>":CLOSE1
265 A$=INKEY$:IF A$<>CHR$(28) A
ND A$<>CHR$(29) AND A$<>CHR$(31
) AND A$<>CHR$(30) AND A$<>CHR$
(13) THEN 265
266 IF ASC(A$)=30 THEN BT=BT+10
:GOSUB 5000:GOSUB 8000:GOTO 265
267 IF ASC(A$)=31 THEN BT=BT+10
*(BT>9):GOSUB 5000:GOSUB 8000:G
OTO 265
268 IF ASC(A$)=28 THEN BT=BT+1:
GOSUB 5000:GOSUB 8000:GOTO 265
269 IF ASC(A$)=29 THEN BT=BT+1*
(BT>0):GOSUB 5000:GOSUB 8000:GO
TO 265
270 GOSUB 5800:BT=INT(BT):IF BT
<1 OR BT>MN THEN 240
280 OB=BT:MN=MN-BT
290 CX=63:GOSUB 3000
300 D2=N:GOSUB 1000:LINE (61,10
6)-(105,185),12,BF:FOR I=107 TO
 180 STEP 3:LINE (62,I)-(105,I+
2),6-8*(I/2=INT(I/2)),BF:NEXT
320 IF PL=11 AND PA=1 AND NC=2
THEN 650
330 CX=CX+32
340 PT=PL+10*(PA AND(PL<12))
350 IF PL>21 THEN 680
360 IF NC=5 THEN GOSUB 5000:GOS
UB 6000:PRINT#1, "VOCE TEM UMA
MAO DE CINCO":P5=1:GOSUB 5500:G
OTO 500
370 GOSUB 5000:GOSUB 6000
380 GOSUB 8000
390 GOSUB 5000:GOSUB 7000
400 IF TW=0 THEN GOSUB5000:PRES
ET(8,80):PRINT#1,"(C)ompra ou":
CLOSE1
410 GOSUB 5000:PRESET (8-12*8*(
TW=0),80):PRINT#1,"(P)ede mais
[illegible] acha":PRESET(8,92):PRINT#1,"
(S)uficiente ?"
430 A$=INKEY$:IF A$<>"P" AND (A
$<>"S" OR PT<16) AND (A$<>"C" O
R TW=1) THEN 430
440 IF A$="S" THEN GOSUB 5800:G
OTO 490
450 IF A$="P" THEN TW=1:GOSUB 5
800:GOTO 480
460 IF MN<OB THEN GOSUB 5800:GO
SUB 5000:PRESET(8,80):PRINT#1,"
Você não tem":PRESET(8,92):PRIN
T#1,"tanto dinheiro assim":GOSU
B 5500:GOTO 400
470 BT=BT+OB:MN=MN-OB:GOSUB 700
0:GOSUB 5000:GOSUB 8000:GOSUB [illegible]
000:GOSUB 5800
480 NC=NC+1:GOSUB 3000:GOTO 320
490 IF PT<16 THEN GOSUB 5000:PR
ESET(8,80):PRINT#1,"Você não po
de parar tendo apenas ":PT:GOSU
B 5500:CX=CX-32:GOTO 370
500 GOTO 750
650 GOSUB 5000:GOSUB 6000:PRESE
T(8,88):PRINT#1," VOCE TEM UM N
ATURAL":GOSUB 5500:PF=1:GOTO 86
680 GOSUB 5000:GOSUB 6000:PRESE
T(8,88):PRINT#1,"Você estourou
e perdeu a aposta":GOSUB 5500
750 GOSUB 5000:GOSUB 6000:IF MN
>999 THEN PRINT#1,"PARABENS, VO
CE QUEBROU A BANCA":CLOSE1:PLAY
 "T255ADFBFEADC":GOSUB 5500:GOT
O 790
```

```
760 PRESET(8,80):PRINT#1,"Apert
e <RETURN> para":PRESET(8,92):P
RINT#1,"jogar novamente"
770 IF INKEY$<>CHR$(13) THEN 77
■
780 CLOSE1:GOTO 86
2005 LINE (CX-2,CY-1)-(CX-2,CY+
71),12
3000 GOSUB 1000:GOSUB 2000:IF ■
M>10 THEN PL=PL+10 ELSE PL=PL+N
M
3010 IF NM=1 THEN PA=1
3020 RETURN
5000 OPEN "GRP:" FOR OUTPUT AS
#1:COLOR15:PRESET(8,88):RETURN
5500 CLOSE1:FOR I=1 TO 2000:NEX
T:LINE (0,80)-(255,106),12,BF:R
ETURN
5800 CLOSE1:LINE (0,80)-(255,10
6),12,BF:RETURN
6000 LINE (204,0)-(255,23),12,B
F:PRESET(204,16):PRINT#1,"PONTO
S":PRESET(204,0):PRINT#1,PL;:IF
 PA=1 AND PL<12 THEN PRESET(204
,8):PRINT#1,"OU";PT
6010 RETURN
6500 LINE (204,0)-(255,23),12,B
F:PRESET(204,16):PRINT#1,"PONTO
S":PRESET(204,0):PRINT#1,PL;:IF
 PA=1 THEN PRESET(204,8):PRINT#
1,"OU 11"
6510 RETURN
7000 LINE (204,32)-(255,55),12,
BF:PRESET(204,48):PRINT#1,"FICH
AS":PRESET(204,32):PRINT#1,MN:C
LOSE#1:RETURN
8000 PRESET(204,132):PRINT#1,"A
POSTA":LINE (204,148)-(255,155)
,12,BF:PRESET(204,148):PRINT#1,
BT:CLOSE#1:RETURN
```

A linha ■■ estabelece o valor inicial de uma série de outras variáveis: CX ■ CY são as coordenadas do canto superior esquerdo da carta. **PL** e **DL** são ■ totais do jogador ■ da banca, respectivamente. **NC** é o número de cartas que um deles tem, conforme aquele que esteja jogando no momento. **DA** e **PA** são indicadores que passam a valer 1, caso a banca ou o jogador tenham um ás. **P5** é outro indicador que passa a valer 1, caso o jogador tenha uma "mão de cinco cartas". **TW** mostra se o jogador já pediu uma carta; caso isso tenha acontecido, ■ regras impedem que ele "compre" novas cartas. **PF** indica se o jogador tem um "natural" (21 pontos em duas cartas) e **NA** é ■ número da primeira carta distribuída pela banca.

A sub-rotina que fica entre ■ linhas 3000 ■ 3020 é chamada pela linha 210. Uma carta é mostrada na tela; se ela for maior que 10, o total de pontos do jogador será aumentado em dez pontos; caso contrário, o valor real da carta será somado ao total. A linha 3010 verifica ■ a carta é um ás e modifica o valor de **PA** (indicador de ases do jogador) em caso de necessidade.

A BANCA DÁ AS CARTAS

O programa retorna da sub-rotina 3000 para a linha 220. **D1** é a primeira carta da banca. A sub-rotina 1000 verifica qual o seu valor e naipe. O **FOR...NEXT** desenha o reverso da primeira carta da banca.

Para facilitar a impressão de mensagens na tela de alta resolução, foram feitas várias sub-rotinas no final do programa. A da linha 5000 abre um arquivo "**GRP:**" para que possamos imprimir na tela gráfica por meio de **PRINT 1**; a cor usada é o branco e **PRESET(8,88)** estabelece a posição onde vai ser impressa a mensagem. A sub-rotina da linha 5500 fecha o arquivo aberto pela sub-rotina anterior e apaga a mensagem escrita no meio da tela, usando **LINE,BF**; isso é feito após uma pequena pausa por um laço **FOR...NEXT** para que o jogador tenha tempo de ler a mensagem. A sub-rotina da linha 5800 é igual à da linha 5500, só que não provoca uma pausa.

A sub-rotina da linha 6000 mostra o total de pontos do jogador no canto superior direito da tela. A da linha 6500 exibe, no mesmo local, o valor da primeira carta do jogador. A sub-rotina da linha 7000 mostra o total de fichas que restam ao jogador e a da linha 8000, o valor da aposta feita.

A linha 240 revela os pontos do jogador com o auxílio das sub-rotinas. A linha 250 mostra o valor da aposta — que ainda é zero — e o total de fichas. Estas linhas usam a sub-rotina 5000 para abrir o arquivo que possibilita a impressão na tela gráfica.

A linha 260 solicita ao jogador que faça sua aposta. O arquivo #1 é fechado para evitar problemas — não se deve deixar nunca um arquivo aberto. As linhas 265 a 269 permitem o uso das teclas de controle do cursor para fazer apostas: as setas "direita" e "esquerda" subtraem ou somam 1 ao valor da aposta, as setas "para cima" e "para baixo" somam ou subtraem 10.

A linha 270 apaga a mensagem anterior e também verifica se a aposta foi válida; caso contrário, ela tem que ser feita novamente. Seu valor é subtraído do total de fichas do jogador pela linha 280.

A linha 290 traça a segunda carta do jogador e a linha 300 desenha o reverso da segunda carta da banca. A linha 320 verifica se o jogador tem um "natural" (ou seja, 21 pontos em duas cartas). Neste caso, a linha 650 informa o fato e o jogo recomeça.

A linha 330 cuida da posição da carta seguinte. **PT** é o maior total de pontos que pode ser obtido caso uma das cartas seja um ás. O menor total, ou o único total, se não houver ases, será **PL**. Se este for maior que 21, a linha 350 informará ao jogador que ele "estourou". O programa continua na linha 750, onde são impressas algumas mensagens, conforme a situação.

A linha 360 verifica se o jogador tem uma "mão de cinco". Em caso positivo, o sinalizador **P5** será ativado e o jogador informado; o programa continuará então na linha 500.

A linha 370 mostra os pontos do jogador, a 380 exibe a aposta feita e a 390, o restante das fichas.

AS OPÇÕES DO JOGADOR

As opções de "comprar" ou "pedir" cartas e de "parar" são dadas pelas linhas 400 e 410, enquanto as linhas 430 a 450 cuidam da resposta. Se o jogador tentar "comprar" cartas (o que dobra a aposta) sem ter fichas suficientes, a linha 460 dará a resposta adequada. A linha 470 ajusta e mostra o total de fichas antes que outra carta seja exibida pela linha 480.

Por mais que tente, o jogador não conseguirá "parar" antes de obter pelo menos dezesseis pontos, devido às condições do **IF** da linha 430. A linha 490 é na realidade desnecessária e só entrará em ação caso se mude a condição **PT<16** da linha 430.

A linha 750 atualiza o total de pontos do jogador e verifica se este tem mais de 999 fichas; neste caso, a banca "quebrará" e o jogador será informado, enquanto uma melodia comemora o feito. As linhas 760 e 780 perguntam então se o jogador quer outra "queda".

Apague a linha 170 e acrescente as próximas linhas ao programa do artigo anterior:

```
150 MN = 100
190  HCOLOR= 1: FOR I = 0 TO 19
1: HPLOT 0,I TO 279,I: NEXT
200 CX = 6:CY = 2:PL = 0:DL = 0
:NC = 2:DA = 0:PA = 0:P5 = 0:TW
= 0:PF = 0:NA = N
210  GOSUB 3000
220 D1 = N: GOSUB 1000: FOR I =
 100 TO 180: HCOLOR= 3 + 2 * (
INT (I / 2) = I / 2): HPLOT 6,I
 TO 56,I: NEXT
230  FOR I = 1 TO 3500: NEXT
240  HOME : TEXT : PRINT "VOCE
TEM ";PL;: IF PA = 1 THEN  PRIN
T " OU 11": GOTO 250
245  PRINT
250  PRINT : PRINT "VOCE TEM ";
MN;" FICHAS"
260  PRINT : INPUT "QUANTO VOCE
 APOSTA ?";BT
270 BT =  INT (BT): IF BT < 1 O
R BT > MN THEN 240
280 OB = BT:MN = MN - BT
290 CX = 60: GOSUB 3000
300 D2 = N: GOSUB 1000: FOR I =
 100 TO 180: HCOLOR= 3 + 2 * (
INT (I / 2) = I / 2): HPLOT 60,
I TO 110,I: NEXT
310  FOR K = 1 TO 4000: NEXT
320  IF PL = 11 AND PA = 1 AND
NC = 2 THEN 650
330 CX = CX + 54
340 PT = PL + 10 * (PA AND (PL
< 12))
350  IF PL > 21 THEN 680
360  HOME : TEXT : IF NC = 5 TH
EN  PRINT "VOCE TEM UMA MAO DE
CINCO":P5 = 1: FOR K = 1 TO 250
0: NEXT : GOTO 500
370  PRINT "VOCE TEM ";PL;: IF
PA = 1 AND PL < 12 THEN  PRINT
" OU ";PT: GOTO 380
375  PRINT
380  PRINT : PRINT "VOCE APOSTO
U ";BT
390  PRINT : PRINT "VOCE TEM ";
MN;" FICHAS"
400  PRINT : IF TW = 0 THEN  PR
INT "(C)OMPRA OU "
410  PRINT : PRINT "(P)EDE MAIS
 ";: IF PT > 15 THEN  PRINT "OU
 ACHA (S)UFICIENTE";
420  PRINT " ?"
430  GET A$: IF A$ <  > "P" AND
 (A$ <  > "S" OR PT < 16) AND (
A$ <  > "C" OR TW = 1) THEN 430
440  IF A$ = "S" THEN 490
450  IF A$ = "P" THEN TW = 1: G
OTO 480
460  IF MN < OB THEN  PRINT : P
RINT "VOCE NAO TEM TANTO DINHEI
RO ASSIM": GOTO 400
470 BT = BT + OB:MN = MN - OB
480 NC = NC + 1: GOSUB 3000: GO
TO 310
490  IF PT < 16 THEN  PRINT : P
RINT "VOCE NAO PODE PARAR TENDO
 APENAS ";PT:CX = CX - 50: GOTO
 370
500  GOTO 750
650  HOME : TEXT : PRINT "VOCE
TEM UM NATURAL": FOR K = 1 TO 1
500: NEXT :PF = 1: GOTO 190
680  HOME : TEXT : PRINT "VOCE
ESTOUROU E PERDEU A APOSTA"
750  PRINT : PRINT "VOCE TEM ";
MN;" FICHAS": PRINT : IF MN > 9
99 THEN  HOME : INVERSE : HTAB
8: VTAB 11: PRINT "VOCE QUEBROU
 A BANCA": FOR I = 1 TO 2000:XX
 =  PEEK ( - 16336): NEXT : GOT
O 790
760  PRINT : PRINT "APERTE <RET
URN> PARA OUTRA MAO"
770  GET A$: IF A$ <  >  CHR$ (
13) THEN 770
780  GOTO 190
3000  POKE  - 16299,0: POKE  -
```

```
16304,0: GOSUB 1000: GOSUB 2000
: IF NM > 10 THEN PL = PL + 10:
 GOTO 3010
3005 PL = PL + NM
3010  IF NM = 1 THEN PA = 1
3020  RETURN
```

Os leitores que possuem vídeo monocromático e que no último artigo preferiram apagar a linha 170 devem substituir a linha 190 por:

```
190 HGR2
```

A linha 150 faz com que o número de fichas (**MN**) do jogador seja 100. A linha 190 limpa a tela.

A linha 200 estabelece os valores iniciais de uma série de variáveis. CX e CY são as coordenadas do canto superior esquerdo da carta. **PL** a **DL** são os totais de pontos do jogador e da banca, respectivamente. **NC** é o número de cartas do jogador ou da banca, conforme aquele que estiver jogando. **DA** e **PA** são indicadores, respectivamente, de que o jogador ou a banca possuem um ás. **P5** é um indicador de que o jogador possui uma "mão de cinco cartas". **TW** revela se o jogador já "pediu" cartas, sendo então utilizado para evitar que ele "compre" cartas, segundo as regras do jogo. **PF** é um indicador de "naturais" (ou seja, pontos em duas cartas). **NA** é o número da primeira carta dada pela banca. Um indicador é uma variável que vale 0 ou 1, de acordo com a situação.

A sub-rotina da linha 3000 é chamada pela linha 210. Uma carta é colocada na tela; se for maior que 10, o total de pontos do jogador será aumentado em 10; caso contrário, o valor da carta será somado aos pontos do jogador. A linha 3010 verifica se a carta é um ás, se necessário.

A BANCA DÁ AS CARTAS

O programa retorna da sub-rotina para a linha 220. Ali, **D1** é a primeira carta da banca e a sub-rotina 1000 é usada para descobrir seu valor e naipe. O **FOR...NEXT** desenha o reverso da carta da banca.

A linha 230 faz uma pausa antes que a linha 240 limpe e ative a tela de textos e informe o total ao jogador. A mensagem adicional "OU 11" é usada somente quando a carta é um ás. O número de fichas é escrito pela linha 250. A linha 260 cuida da aposta do jogador e a 270 providencia para que a aposta seja maior que zero e esteja dentro das posses do jogador (caso contrário, ela terá que ser feita novamente). O valor da aposta é subtraído das fichas do jogador pela linha 280.

A seguir, a linha 290 chama a sub-rotina que distribui as cartas, e a linha 300 desenha a parte posterior da segunda carta da banca.

A linha 320 verifica se o jogador tem um "natural" depois da pausa causada pela linha 310. Em caso positivo, a linha 650 informará o fato ao jogador e o programa recomeçará.

A linha 330 cuida da posição da carta seguinte. **PT** é o maior dos totais de pontos, para o caso de o jogador possuir um ás. Se não houver ases, **PL** será o menor (ou o único) total. Se **PL** for maior que 21, a linha 680 informará ao jogador que ele "estourou". O programa continuará, então, imprimindo as mensagens da linha 750 em diante.

A linha 360 verifica se o jogador tem uma "mão de cinco cartas", colocando 1 em **P5** e criando uma pausa, caso isto seja necessário.

A linha 370 mostra o total contido nas cartas do jogador (dois valores, se houver um ás e o valor menor for inferior a 12). A aposta do jogador é impressa pela linha 380 e a quantidade de fichas pela linha 390.

AS OPÇÕES DO JOGADOR

As opções de "parar", "comprar" ou "pedir" cartas são oferecidas pelas linhas 400 a 420. As linhas 430 a 450 cuidam das respostas do jogador. Se este tentar "comprar" cartas sem ter dinheiro para dobrar a aposta, a linha 460 tratará de informá-lo. A linha 470 acerta o valor da aposta e do total de fichas antes da linha 480 dar a próxima carta.

A condição **PT < 16** da linha 430 impede que o jogador "pare" sem ter pelo menos dezesseis pontos. A linha 490 é desnecessária e só entrará em ação se essa condição for modificada. O programa continua na linha 750, que avisa ao jogador o número de fichas restantes, verificando se a banca foi "quebrada". Se isso acontecer, um ruído será produzido.

As linhas 760 a 780 oferecem ao jogador a opção de jogar outra "mão".

[TK-2000]

Para que o programa funcione no TK-2000 são necessárias algumas modificações. Em primeiro lugar, daqui para a frente use apenas a HGR2 (segunda página de vídeo). Quando o programa estiver completo, qualquer retorno a **MA** (primeira página de vídeo) vai danificá-lo. Então, primeiro digite **MP** e depois **<RETURN>**; carregue o programa inicial e faça as mesmas modificações sugeridas para o Apple com exceção das seguintes linhas:

```
160  HGR2
240  GOSUB 5000: PRINT "VOCE TE
M ";PL;: IF PA = 1 THEN  PRINT
" OU 11": GOSUB 6000: GOTO 250
245  PRINT : GOSUB 6000
250  GOSUB 5000: PRINT "VOCE TE
M ";MN;" FICHAS": GOSUB 6000
260  GOSUB 5000: INPUT "QUANTO
VOCE APOSTA ?";BT: GOSUB 6000
360  GOSUB 5000: IF NC = 5 THEN
  PRINT "VOCE TEM UMA MAO DE CI
NCO":P5 = 1: GOSUB 6000 GOTO 50
0
370  PRINT "VOCE TEM ";PL;: IF
PA = 1 AND PL < 12 THEN  PRINT
" OU ";PT: GOSUB 6000: GOTO 380
375  PRINT : GOSUB 6000
380  GOSUB 5000: PRINT "VOCE AP
OSTOU ";BT: GOSUB 6000
390  GOSUB 5000: PRINT "VOCE TE
M ";MN;" FICHAS": GOSUB 6000
400  GOSUB 5000: IF TW = 0 THEN
  PRINT "(C)OMPRA OU "
460  IF MN < OB THEN  GOSUB 500
0: PRINT "VOCE NAO TEM TANTO DI
NHEIRO ASSIM": GOSUB 6000: GOTO
 400
490  IF PT < 16 THEN  GOSUB 500
0: PRINT "VOCE NAO PODE PARAR T
ENDO APENAS ";PT:CX = CX - 50:
GOSUB 6000: GOTO 370
650  GOSUB 5000: PRINT "VOCE TE
M UM NATURAL": GOSUB 6000:PF =
1
660  GOSUB 4000: GOSUB 5000: IF
 DL = 11 THEN  PRINT "MAS A BAN
CA TEM A MESMA COISA": GOSUB 60
00: GOTO 610
680  GOSUB 5000: PRINT "VOCE ES
TOUROU E PERDEU A APOSTA": GOSU
B 6000
690  GOSUB 5000: IF MN < 1 THEN
  PRINT "VOCE PERDEU TODAS AS F
ICHAS": GOSUB 6000: GOTO 790.
700  IF PF = 1 THEN  PRINT "EMB
ARALHANDO AS CARTAS": GOSUB 150
0: GOSUB 6000: GOTO 750
750  GOSUB 5000: PRINT "VOCE TE
M ";MN;" FICHAS": GOSUB 6000: I
F MN > 999 THEN  GOSUB 5000: PR
INT "VOCE QUEBROU A BANCA": FOR
 I = 1 TO 2000: NEXT : GOTO 790
760  GOSUB 5000: PRINT "APERTE
<RETURN> PARA OUTRA MAO": GOSUB
 6000
5000  HCOLOR= 1: FOR I = 84 TO
108: HPLOT 0,I TO 255,I: NEXT :
 HTAB 1: VTAB 12: RETURN
6000  FOR I = 1 TO 1000: NEXT :
 RETURN
```

Note que você terá que procurar na listagem das modificações para o Apple algumas linhas que não estão aqui por falta de espaço. A diferença entre o programa do Apple e do TK-2000 se deve ao fato de este último não possuir uma página só para textos. Assim, as men-

sagens terão que ser impressas ■ tela gráfica.

A sub-rotina da linha 5000 apaga qualquer coisa que tenha sido escrita antes e posiciona o cursor para a mensagem seguinte. A sub-rotina 6000 provoca uma pausa para que ■ mensagem possa ser lida. No restante, o programa é igual ao do Apple ■ as explicações são as mesmas.

As linhas 220 e 300 são um pouco diferentes das do Apple. Elas colocam as cartas da banca mais abaixo, abrindo ■ espaço para que as mensagens sejam impressas na tela.

T

Acrescente as próximas linhas àquelas que digitou da outra vez:

```
170 MN=100
190 PCLS 6
200 CX=6:CY=11:PL=0:DL=0:NC=2:D
A=0:PA=0:P5=0:TW=0:PF=0:NA=N
210 GOSUB 3000
220 POKE 178,156:D1=N:GOSUB 100
0:LINE (6,108)-(50,180),PSET,BF
230 FOR K=1 TO 2000:NEXT
240 CLS:PRINT" VOCE TEM";PL;:IF
 PA=1 THEN PRINT "OU 11" ELSE P
RINT
250 PRINT:PRINT" VOCE TEM";MN;"
FICHAS"
260 INPUT" FACA SUA APOSTA ";BT
270 BT=INT(BT):IF BT<1 OR BT>MN
 THEN 240
280 OB=BT:MN=MN-BT
290 CX=56:GOSUB 3000:POKE 178,1
56
300 D2=N:GOSUB 1000:LINE(56,108
)-(100,180),PSET,BF
310 FOR K=1 TO 2500:NEXT
320 IF PL=11 AND PA=1 AND NC=2
THEN 650
330 CX=CX+50
340 PT=PL+10*(PA AND(PL<12))
350 IF PL>21 THEN 680
360 CLS:IF NC=5 THEN PRINT" VOC
E TEM ■ MAO DE CINCO":P5=1:FO
R K=1 TO 1500:NEXT:GOTO 500
370 PRINT" VOCE TEM";PL;:IF PA=
1 AND PL<12 THEN PRINT "OU";PT
ELSE PRINT
380 PRINT:PRINT" VOCE APOSTOU";
BT
390 PRINT:PRINT" VOCE TEM";MN;"
FICHAS"
400 PRINT:IF TW=0 THEN PRINT" (
C)OMPRA OU";
410 PRINT" (P)EDE MAIS CARTAS";
:IF PT>15 THEN PRINT" OU (S)ATI
SFEITO";
420 PRINT " ?"
430 A$=INKEY$:IF A$<>"P" AND (A
$<>"S" OR PT<16) ■ (A$<>"C" O
R TW=1) THEN 430
440 IF A$="S" THEN 490
450 IF A$="P" THEN TW=1:GOTO 48
0
460 IF MN<OB THEN PRINT " VOCE
NAO TEM TANTO DINHEIRO":GOTO 40
0
470 BT=BT+OB:MN=MN-OB
480 NC=NC+1:GOSUB 3000:GOTO 310
490 IF PT<16 THEN PRINT " VOCE
NAO PODE PARAR AGORA":PT:CX=CX-
50:GOTO 370
500 GOTO 750
650 CLS:PRINT" VOCE TEM UM NATU
RAL":FOR K=1 TO 1000:NEXT:PF=1:
GOTO 190
680 CLS:PRINT" VOCE ESTOUROU ■
PERDEU A APOSTA"
750 PRINT:PRINT" VOCE TEM";MN;"
FICHAS":PRINT:IF MN>999 THEN PR
INT "MUITO BEM, VOCE QUEBROU A
BANCA!":SCREEN 0,1:PLAY"T6ADFBF
EADC":GOTO 790
760 PRINT:PRINT" PRESSIONE (S)
PARA RECOMECAR"
770 IF INKEY$<>"S" THEN 770
780 GOTO 190
3000 SCREEN 1,1:GOSUB 1000:GOSU
B 2000:IF NM>10 THEN PL=PL+10 E
LSE PL=PL+NM
3010 IF NM=1 THEN PA=1
3020 RETURN
```

A linha 170 faz com que o número de fichas do jogador seja igual ■ cem. A linha 190 limpa ■ dá cor à tela.

A linha 200 estabelece o valor inicial de uma série de variáveis. CX ■ CY são as coordenadas do canto superior esquerdo da carta. **PL** e **DL** são, respectivamente, os totais de pontos do jogador e da banca. **NC** é o número de cartas de quem estiver jogando no momento, jogador ou banca. **DA** e **PA** são indicadores que passam a valer 1 se ■ banca ou o jogador tiverem ■ ás. **P5** é um indicador que sofrerá a mesma modificação ■ o jogador consiga uma "mão de cinco cartas". **TW** é um indicador que passará a valer 1 quando o jogador "pedir" uma carta. Segundo as regras, a partir de então ele não poderá mais "comprar". **PF** é um indicador de "naturais" e **NA** o número da primeira carta dada pela banca.

A sub-rotina da linha 3000 é chamada pela linha 210. Uma carta é colocada no vídeo. Se for maior que 10, o total do jogador será aumentado em dez; caso contrário, o valor da carta será somado aos pontos do jogador. Se ■ carta for um ás, a linha 3010 colocará 1 no sinalizador de ases **PA**.

A BANCA DÁ AS CARTAS

Quando o programa retornar, a linha 220 usará **POKE** para obter o padrão usado no reverso das cartas. **D1** é a primeira carta da banca; a sub-rotina descobre seu valor e naipe. **LINE** desenha ■ carta e **POKE** cria o padrão.

A linha 230 provoca uma pausa antes que ■ linha 240 limpe a tela ■ escreva o total do jogador. O comando **PRINT** muda automaticamente a tela gráfica para a tela de textos. Se a carta for um ás, aparecerá uma mensagem adicional "OU 11". O número de fichas é informado pela linha 250.

A linha 260 cuida da aposta; ■ linha 270 verifica se esta é válida; caso contrário, ela deve ser refeita. A aposta é subtraída das fichas do jogador pela linha 280.

A seguir, a linha 290 chama as subrotinas que dão as cartas ■ a linha 300 desenha a segunda carta da banca.

A linha 320 verifica se o jogador tem um "natural" (isto é, 21 pontos em duas cartas), depois da pausa criada pela linha 310. Se houver um "natural", a linha 650 informará ■ jogador, e o jogo começará de novo.

A linha 330 cuida da posição da carta seguinte. Se o jogador tiver um ás, o maior dos seus totais será **PT**. Se não houver ases, o menor (ou único) total será **PL**. Se este for maior que 21, a linha 680 informará ao jogador que houve um "estouro", e o programa irá para a linha 750. A linha 350 verifica se o jogador tem uma "mão de cinco". Neste caso, o indicador **PS** será ativado e uma pausa será provocada antes de ■ jogo prosseguir.

A linha 370 mostra o total dos pontos do jogador — dois valores, se houver um ás e o valor menor for inferior a 12. A aposta é mostrada pela linha 380 e as fichas remanescentes são indicadas pela linha 390.

AS OPÇÕES DO JOGADOR

As opções de "parar", "comprar" ou "pedir" cartas são oferecidas ao jogador pelas linhas 400 a 420. As linhas 430 a 450 cuidam das respostas. Se o jogador tentar comprar cartas sem ter fichas suficientes para dobrar a aposta, a linha 460 dará ■ aviso. A linha 470 atualiza o valor da aposta e do total de fichas, antes que ■ linha 480 dê outra carta.

Se o jogador tentar "parar" sem ter pelo menos dezesseis pontos, será impedido pela condição **PT < 16** na linha 430. A linha 490 é desnecessária e só entrará em ação caso ■ condição seja modificada.

A linha 750 diz ao jogador quantas fichas sobraram e verifica se ele conseguiu quebrar a banca. Se isso acontecer, haverá um efeito sonoro depois de uma mudança de cores.

As linhas 760 a 780 permitem ■ jogador uma outra "mão".

COMO COMBINAR PROGRAMAS

Uma das características da programação consiste em que tudo o que não esteja gravado em fita ou no disquete tem que ser digitado. Freqüentemente, contudo, pode-se economizar tempo, modificando programas já prontos ou combinando dois (ou mais) programas para formar um terceiro.

Existem basicamente duas maneiras de combinar programas. A primeira consiste em somar duas partes com números de linhas diferentes. A segunda, mais complexa, permite a união de programas com números de linhas iguais. Neste curso, os dois métodos serão tratados simultaneamente.

QUANDO COMBINAR

Combinar programas (*Merge* é o termo usado em inglês) é essencialmente um método de auxílio à programação. Suponhamos, por exemplo, que você tenha feito um programa que não funciona do modo esperado, ou que deseje alterá-lo para que ele execute uma tarefa diferente daquela para a qual foi criado. A melhor maneira seria gravá-lo em fita ou disquete e continuar a trabalhar nele sem medo de danificar o original. Uma vez aperfeiçoado, o programa estaria, digamos, com o dobro do tamanho original, ou teria ganho vários módulos diferentes. Em qualquer caso, as duas versões podem ter elementos que você deseje conservar, mas existem entre elas tantas diferenças que seria muito difícil digitar cada coisa separadamente. E há mais um problema: a não ser que você coloque os dois programas no papel, é necessário ter tudo na memória para poder fazer as mudanças na hora de combiná-los.

Outra situação em que se torna necessário combinar programas é aquela em que se deseja incorporar rotinas ou procedimentos já existentes a um novo programa. Estes podem ser longas rotinas em código de máquina que tocam uma música, ou uma rotina que anima um desenho em alta resolução, ou mesmo uma rotina de detecção de erros.

A maioria dos programadores tem uma variada "biblioteca" de sub-rotinas, montadas ao longo dos anos e incorporadas aos novos programas de acordo com as necessidades. Uma "biblioteca" desse tipo tem um valor inestimável pois, uma vez que as rotinas tenham sido digitadas, testadas e gravadas, não é necessário nem mesmo desejável redigitá-las toda vez que for preciso utilizá-las. Elas podem ser incorporadas ao programa por meio das técnicas de combinação.

Esse recurso serve igualmente para enfileirar uma série de pequenos programas e executá-los seqüencialmente. Neste caso, pode-se ter, por exemplo, uma série que comece por uma saudação inicial, seguida de um desenho multicolorido, e assim por diante.

A última linha pode desviar o programa para o início, provocando uma execução contínua.

Por que perder tempo digitando rotinas e programas já digitados e testados, ■ o computador pode fazer todo ■ trabalho com a ajuda apenas de alguns comandos?

- POR QUE JUNTAR PROGRAMAS
- COMO ACRESCENTAR SUB-ROTINAS PRÉ-FABRICADAS
- COMO COMBINAR VÁRIOS PROGRAMAS EM UM SÓ

COMO COMBINAR

Cada computador tem seu próprio método para fazer a combinação de programas. Alguns dispõem de um comando (**MERGE**) que se encarrega da tarefa, fazendo com que o programa que está na memória seja retido enquanto outro é carregado da fita ou disquete. Já outros exigem que se digite algumas linhas de instrução, ■ se destine espaço ■ memória para o segundo programa.

Qualquer que seja o método utilizado, no entanto, é necessário que os números de linhas sejam ajustados antes para que ■ combinação se dê da forma desejada.

O ZX-81 é ■ caso especial. Ele não tem um comando para fazer ■ combinação e não há nenhuma técnica razoavelmente simples que permita que dois programas sejam carregados na memória. Veja a seção *Perguntas e respostas* para maiores informações.

5

No Spectrum, a combinação de programas é feita por intermédio do comando **MERGE**, que torna muito simples essa operação. No entanto, é necessário atenção com a numeração dos programas a serem combinados. Por exemplo, para juntar uma sub-rotina de umas vinte linhas a um programa maior, é preciso deixar um espaço na numeração. Assim, o programa seria numerado de 10 a 50, digamos, e de 300 a 1000, e a sub-rotina de 60 a 250.

Outra possibilidade é dar à sub-rotina uma numeração superior à do programa. Com um pouco de cuidado, pode-se arranjar as coisas de tal forma que, com a substituição de algumas linhas por outras, e a incorporação de novas linhas, obtenha-se um novo programa.

Agora, digite e execute o programa ■ seguir. Ele mostra quadrados coloridos em posições aleatórias. Mais adiante, você verá como combiná-lo com outro programa:

```
 10 BORDER 0: INK 9
 50 PAPER 0
 60 CLS
 70 FOR n=1 TO 400
 80 LET x=INT (RND*32)
 90 LET y=INT (RND*22)
100 PAPER 7: IF x>8 ■ x<24
■ y>6 AND y<16 THEN  PAPER ■
110 PRINT AT y,x;" "
120 NEXT n
```

Grave o programa (com o título de SQ1, por exemplo). E não desconecte o gravador — você ainda vai precisar dele. Note que qualquer outro programa que você já tenha serviria para nossa demonstração. Digite **NEW** ■ em seguida ■ próximo programa:

```
 10 BORDER 0: PAPER 0: ■ 9:
CLS
 30 FOR t=1 TO 5
 40 INK INT (RND*6)+2
 60 PRINT AT t,0;"ISTO ■ UM TE
STE"
130 NEXT t
```

Existe uma forma simples de combinar programas no ZX-81?

Infelizmente, não. Isso pode ser feito em linguagem de máquina, mas o programa é bem complicado.

O ZX-81 não tem um comando MERGE e não existe nenhuma forma em BASIC de carregar dois programas na memória ao mesmo tempo (todos os programas são carregados no mesmo endereço da memória). Assim, ao se carregar um programa, inevitavelmente se destrói o anterior.

Este é um programa muito simples, que mostra uma mensagem (linha 60) cinco vezes. Agora digite **MERGE""** para combiná-lo com o anterior (não se esqueça de retroceder a fita até o início do programa). O programa da fita será então adicionado ao da memória do micro. Liste o novo programa e observe que as linhas 10 e 60 foram substituídas, enquanto as linhas 50 e de 70 a 120 foram acrescentadas. Execute o programa para ver o procedimento inicial (SQ1) executado cinco vezes.

Como não é muito fácil renumerar programas no Spectrum, convém ajustar a numeração das linhas antes de combinar os programas.

T

O TRS-Color permite a junção de programas em seqüência, mas não a sobreposição de linhas.

Digite e execute o programa seguinte, que coloca quadradinhos coloridos em áreas retangulares da tela.

```
10 PCLEAR 4
20 FOR T=1 TO 5
30 CLS 0
40 FOR N=1 TO 400
50 X=RND(32)-1
60 Y=RND(16)-1
70 IF X<6 OR X>25 OR Y<4 OR Y>1
1 THEN C=175 ELSE C=255
80 POKE 1024+Y*32+X,C
90 NEXT
100 NEXT
```

Agora, grave o programa (como SQ1, por exemplo), para que ele possa ser carregado novamente na memória. Ele poderia ser uma longa sub-rotina ou parte de um programa inacabado.

Digite o comando **NEW** e, em seguida, o próximo programa:

```
10 PCLEAR 4
20 CLS
30 PRINT @192,STRING$(32,CHR$(2
36));
40 PRINT " BENVINDO A UM DISPLA
Y COLORIDO"
50 PRINT STRING$(32,CHR$(163))
```

O programa imprime uma mensagem na tela (linha 40). Para juntar a ele o programa que você já gravou, entre as linhas a seguir:

```
POKE 25,PEEK(27)
POKE 26,PEEK(28)-2
```

Em seguida, carregue o programa da fita (SQ1) e entre estas instruções:

```
POKE 25,30
POKE 26,1
```

Ao listar o novo programa, ele aparecerá com números de linha de 10 a 50 e a seguir de 10 a 100. Digite **RENUM** para renumerar o programa e liste-o novamente; você verá algo como:

```
10 PCLEAR 4
20 CLS
30 PRINT @192,STRING$(32,CHR$(2
36));
40 PRINT " BENVINDO A UM DISPLA
Y COLORIDO"
50 PRINT STRING$(32,CHR$(163))
60 PCLEAR 4
70 FOR T=1 TO 5
80 CLS 0
90 FOR N=1 TO 400
100 X=RND(32)-1
110 Y=RND(16)-1
120 IF X<6 OR X>25 OR Y<4 OR Y>
11 THEN C=175 ELSE C=255
130 POKE 1024+Y*32+X,C
140 NEXT
150 NEXT
```

A primeira parte do programa é executada rapidamente. Para separá-la da segunda parte, mude a linha 60 para:

```
60 FOR K=1 TO 2000:NEXT
```

MSX

O MSX é outro computador que apresenta grande facilidade para combinação de programas. Como o Spectrum, ele possui o comando **MERGE**, que se encarrega de todo o trabalho.

Todavia, é sempre importante tomar cuidado com a numeração das linhas, pois, se existirem linhas de mesmo número nos dois programas a serem unidos, aquelas que vêm do programa que é carregado da fita substituirão as da memória. Assim, deve-se sempre deixar espaço suficiente no programa que vai receber o novo trecho para que não se percam linhas.

Digite o programa apresentado a seguir. Ele desenhará quadradinhos coloridos em determinada região da tela. Em seguida grave-o na fita, usando o comando **SAVE** (e não **CSAVE**). Chame-o, por exemplo, de SQ1.

```
60 R=RND(-TIME)
70 SCREEN3
80 COLOR 15,R*15
90 FORI=1TO400
100 X=INT(RND(1)*256)
110 Y=INT(RND(1)*192)
120 IFX<36ORX>210ORY<20ORY>110T
HEN100
130 PSET(X,Y),RND(1)*15
140 NEXT
```

Quando tiver terminado a gravação do programa, digite **NEW** e, sem desconectar o gravador do micro, coloque este outro programa no seu computador:

```
10 CLS
20 PRINTSTRING$(39,36)
30 PRINTTAB(10);"VEJA QUE BELO
EFEITO!"
40 PRINT:PRINTSTRING$(39,36)
50 FORJ=1TO500:NEXT
150 SCREEN0
160 PRINTSTRING$(39,123)
170 PRINTTAB(10);"ACABOU!!!"
```

Este é um programa muito simples, que coloca duas mensagens na tela. Sozinho, não quer dizer muita coisa; mais adiante, porém, você poderá avaliar sua importância.

Agora, rebobine a fita para posicioná-la no início do programa anterior. Digite **MERGE"SOL"**. Em seguida, digite **LIST**. Veja que os dois programas estão juntos, o primeiro colocado dentro do segundo.

Como o MSX também oferece o comando **RENUM**, você não terá dificuldade para fazer com que os números das linhas dos programas sejam os mais adequados às combinações que quiser fazer.

O Apple II não tem o comando **MERGE** no BASIC; mas, para quem trabalha com uma unidade de disco, existe um programa no disco-mestre do sistema operacional que renumera as linhas de um programa em BASIC e permite que se faça a combinação de programas. Para trabalhar com ele, basta seguir as instruções do manual.

Aqui, mostraremos duas outras formas de se combinar programas. A primeira é destinada a quem trabalha com disquetes. Essa técnica aproveita a facilidade do Apple para ler um arquivo texto, simulando entradas de teclado. Pode-se guardar, assim, uma série de instruções (uma rotina de ordenação, por exemplo) na forma de um arquivo

texto e, por meio do comando **EXEC**, fazer com que ■ rotina seja adicionada ao programa da memória. Tudo funciona como ■ a rotina tivesse sido digitada no teclado. Vejamos como isso acontece. Digite o programa ■ seguir:

```
1 ■ =  CHR$ (4)
2  PRINT D$;"OPEN TEXT"
3  PRINT D$;"WRITE TEXT"
4  POKE 33,33: LIST 10 -
5  PRINT D$;"CLOSE": POKE 33,40
: END
100 ■ =  CHR$ (4)
110  PRINT D$;"OPEN TEXT"
120  PRINT D$;"WRITE TEXT"
130  POKE 33,33: LIST  - 5
140  PRINT D$;"CLOSE": POKE 33,
40
```

Depois de digitá-lo, execute-o a partir da linha 100 (**RUN 100**). Isto fará com que seja criado ■ arquivo texto no seu disco que contém as linhas 1 ■ 5. Ao verificar o diretório (catálogo) do disco (**CATALOG**), você verá que ■ arquivo tem o nome de **TEXT**. Modifique-o para **MERGE** e você ■ lembrará facilmente do nome (**RENAME TEXT, MERGE**). Agora digite **NEW** e a seguir o próximo programa:

```
40  GR
50  ■ I = 1 TO 500
60  COLOR=  RND (1) * 16
■   PLOT X,Y
100  NEXT
70 ■ =  RND (1) * ■
80 ■ =  ■ (1) * 40
```

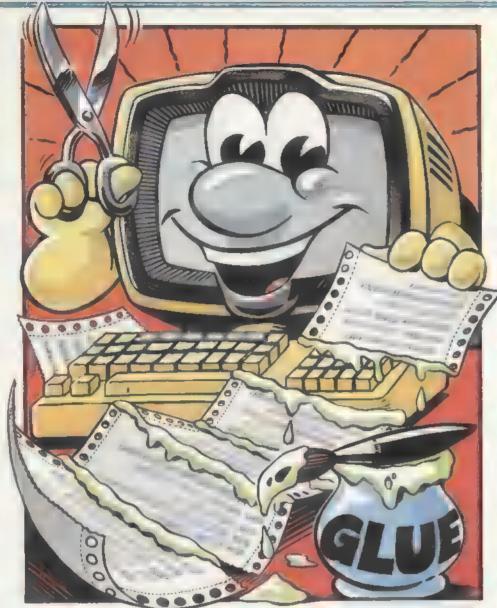
O programa mostra ■ grande quantidade de quadrados coloridos ■ tela; é bem simples, ■ servirá para ■ nossa demonstração. Agora digite **MON C,I,O** ■ depois **EXEC MERGE**. Você verá ■ linhas 1 a ■ do programa anterior apa■ na tela. Liste o programa e verifique se as linhas que você havia digitado ainda estão lá. Dessa forma, você já conseguiu combinar dois programas: ■ que desenha ■ tela e o outro, que pode não lhe parecer muito claro. Agora, ■ explicá-lo melhor. A primeira linha apenas define ■ variável ■ como ■ caractere de código 4; este indica ao computador que um comando do siste■ operacional está sendo usado. A seguir, ■ arquivo é aberto ■ preparado para receber dados (linhas ■ e 3). A linha 4 coloca a listagem do programa nesse arquivo, ■ partir da linha 10. O arquivo é então fechado ■ termina ■ rotina.

A vantagem dessa operação consiste em que uma rotina assim armazenada pode ser adicionada a qualquer progra■ com apenas um comando.

Para compreender melhor, digite **RUN**. Nossa pequena rotina será executada e ■ programa que coloca os quadrados coloridos na tela será armazenado no arquivo de nome TEXT. À medida que outras rotinas forem sendo guardadas, é necessário dar a elas novos nomes como foi feito com **MERGE**. Agora digite **NEW** e a seguir o próximo programa:

```
10  HOME
20  PRINT "***    DEMONSTRACAO
DO ■        ***"
30  FOR ■ = 1 TO 2000: NEXT
110  TEXT : GOTO 10
```

Este programa imprime apenas duas mensagens na tela. Para uni-lo ao programa dos quadradinhos, você deve digitar **EXEC TEXT**. Feito isso, ■ dois programas podem ser executados como se fossem um só.

Se ■ ■ micro tem um comando para a renumeração de linhas, use-o para que ■ programa comece em ■ determinada linha ■ tenha um incremento constante. Se você não tem ■ programa, a renumeração deve ■ feita manualmente; mas tome cuidado com todas as linhas referenciadas em instruções **GOTO** ■ **GOSUB** e **ON...GOTO** ■ **ON...GOSUB**.

SEGUNDO MÉTODO

Para quem não tem uma unidade de disco, há um outro método, mais rudimentar, que também permite a combinação de programas. Esse método coloca um programa após o outro, bloqueando ■ mistura de linhas de números variados. Assim, o programa da memória deve ter numeração inferior ao daquele que vai ser carregado da fita.

Digite inicialmente as linhas de 40 ■ 100 do programa anterior. A seguir, grave-o em fita. Limpe ■ memória do micro (NEW) e digite as linhas de 10 ■ 30. Não digite a linha 110. Agora execute as seguintes instruções:

```
E=PEEK(106)*256+PEEK(105)-2
POKE 104,INT(E/256)
POKE 103,E-PEEK(104)*256
```

```
Carregue o programa da fita ■
 o comando LOAD

POKE 104,8
POKE 103,1
LIST
```

E temos ■ dois programas juntos.

T

Os microcomputadores compatíveis com ■ linha TRS-80 que dispõem de BASIC para disco (ou seja, sob o sistema operacional DOS) já têm os comandos **MERGE** — destinado ■ combinar programas (com possibilidade de substituição de linhas ■ de inserção de linhas intermediárias) — e **RENUM**, que permite a renumeração das linhas do programa resultante. Assim, é muito fácil fundir dois ou mais programas em rotinas em BASIC: basta carregar ou digitar o primeiro programa ■ memória (através de um comando **LOAD**) e, em seguida, digitar um comando **MERGE "nome"**, para especificar ■ denominação do programa a ser fundido com o primeiro.

O TRS-80 na versão cassete não tem o comando **MERGE**; para combinar programas nesse micro são necessários, portanto, alguns "truques" especiais. Esses "truques", contudo, permitem apenas que se juntem programas em seqüência, isto é, um depois do outro, impedindo a superposição de linhas.

Digite ■ execute ■ programa seguinte, que coloca uma mensagem várias vezes na tela:

```
100 CLS
110 FOR I=1 TO 5
120 PRINT "GOSTOU ■ TESTE ?"
130 NEXT I
```

Agora, grave ■ programa (como SQ1, por exemplo), para que ele possa ser carregado novamente na memória. Ele poderia ser uma longa sub-rotina ou parte de um programa inacabado.

Digite **NEW** e depois o programa a seguir, que serve para preencher a tela com quadradinhos ■ acaso:

```
10 RANDOM
20 FOR T=1 TO 5
30 CLS
40 FOR N=1 TO 400
50 X=RND(64)-1
60 Y=RND(16)-1
70 SET (X,Y)
80 NEXT N
90 NEXT T
```

Ele imprime uma mensagem na tela. Para juntar ■ programa que você já gravou, entre ■ linha a seguir:

```
PRINT PEEK(16633)
```

Se o resultado mostrado na tela for menor do que 2, digite ■ linhas abaixo:

```
POKE 16548,PEEK(16633)+254
POKE 16549,PEEK(16634)-1
```

Entretanto, ■ o resultado for maior ou igual a 2, digite estas linhas:

```
POKE 16548,PEEK(16633)-2
POKE 16548,PEEK(16634)
```

A seguir, carregue o programa da fita (SQ1), usando o comando **CLOAD**, e entre estas instruções:

```
POKE 16548,233:POKE 16549,66
```

Ao listar ■ novo programa, ele aparecerá com números de linha de 10 ■ 70 e a seguir com os números de 100 em diante. Liste-o novamente, ■ você verá o programa inteiro.

AS REGRAS DO JOGO

- AS REGRAS DO JOGO NORMAL E DA VERSÃO COMPUTADORIZADA
- FUNCIONAMENTO DO PROGRAMA
- AS DECISÕES DA BANCA
- PERDER OU GANHAR

Para completar nosso jogo de vinte-e-um, só falta ensinar o computador a fazer ■ vezes da banca. É hora, portanto, de ■ preparar ■ enfrentá-lo: conheça as regras do jogo.

Antes de passar à última seção do programa, convém conhecer as regras do jogo.

O vinte-e-um utiliza um baralho com 52 cartas. As cartas que vão do 2 ao 10 têm valor igual ao seu número. As figuras valem 10 pontos e o ás pode valer 1 ou 11, conforme a decisão do jogador. Em nosso jogo, o computador calcula os pontos automaticamente.

Joga-se, em geral, com dinheiro ou fichas. Nosso computador é programado para marcar os pontos com fichas. Cabe a ele desempenhar o papel da banca, sendo sempre quem dá as cartas.

No início do jogo, ■ computador embaralha as cartas ■ distribui duas, viradas para baixo ("fechadas"). A carta do jogador aparece "aberta" na tela, mas o programa foi feito de modo que ■ computador não conheça as cartas do jogador. Este faz sua aposta, antes que ele e ■ banca recebam outra carta.

O objetivo do jogo é ter uma mão melhor que a da banca, isto é, um maior número de pontos na soma das cartas. Quem tiver um total maior que 21, estourou, perdendo a aposta. Com um total entre 16 e 21, o jogador só ganha da banca ■ esta possuir um total menor, ou se tiver estourado. Assim, ■ banca tem sempre ■ vantagem do empate.

Existem duas mãos especiais — o "natural": um ás e um dez ou um ás e ■ figura, somando 21 em duas cartas; e a "mão de cinco": uma mão com cinco cartas, cujo total ■ igual ou inferior a 21. Um natural do jogador ganha de qualquer mão da banca, exceto de outro natural. Uma mão de cinco também ganha de qualquer coisa, menos de um natural ou outra mão de cinco.

COMO FUNCIONA O PROGRAMA

Depois que ■ jogador recebeu ■ segunda carta, ■ programa verifica se ele

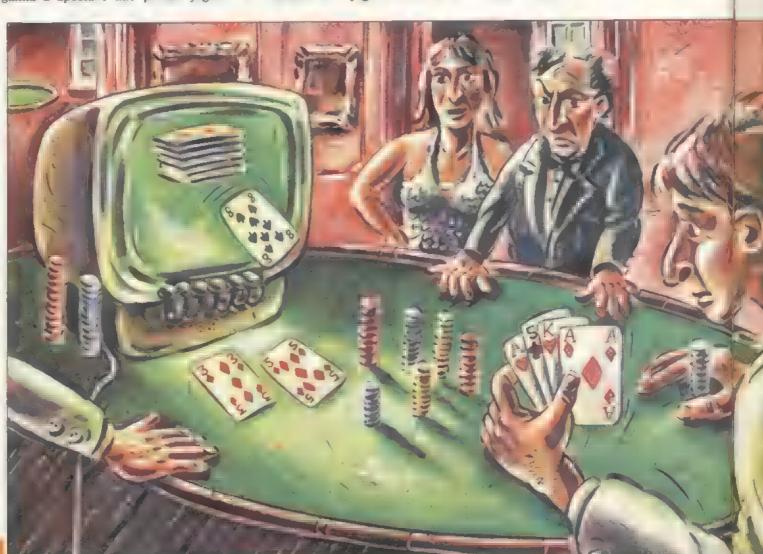
conseguiu um natural. Se não conseguiu, mas obteve ■ total maior ou igual ■ 16, e está satisfeito com este valor, ele pode "parar", não recebendo mais cartas. A vez de jogar passa, então, à banca. Se obteve um total inferior ■ 16, ou não está satisfeito com o valor de sua mão, pode receber mais cartas pelo ato de "pedir" ou de "comprar". No jogo normal, quando se compra uma carta, a aposta original é dobrada e ■ banca dá ■ carta fechada. Na ■ versão computadorizada, não há diferença entre cartas fechadas ou abertas, ■ pedir cartas não dobra a aposta, pois todas são dadas abertas. O jogador não poderá comprar depois de ter pedido alguma carta e, ■ chegar na quinta carta, só poderá pedir. Após cada carta fornecida, ■ computador verifica ■ o jogador ou ■ banca ultrapassaram os 21 pontos.

Quando o jogador estoura, a banca ganha a aposta e não precisa jogar. Quando ele tem 21 ou menos, ■ banca mostra ■ duas cartas ■ decide se pede mais cartas ou pára por ali. Se as duas cartas da banca são um natural, ela ganha ■ aposta. Caso contrário, a banca pede mais cartas até ficar satisfeita com ■ total, conseguir uma mão de cinco ou então estourar — para alegria do jogador.

Se ■ banca pára com um total menor que 21 e sem ter obtido uma mão de cinco, surge na tela a mensagem: a BANCA PAGA... totais que tenham ■ ponto ■ mais que ■ valor de sua mão. Se ■ mão da banca for 21, ■ mensagem NATURAIS ■ MÃOS DE CINCO APENAS ■ mostrada. E, caso obtenha uma mão de cinco, o jogador é informado: APENAS NATURAIS. No jogo convencional, cada jogador declararia o valor de sua mão, se tivesse vencido ■ banca. Na nossa versão, ■ computador soma as cartas do jogador para saber seu total. Se ■ jogador tiver um total maior ou igual ao que ■ banca paga, ele vence; se tiver menos, perde. Quando vence, ■ banca paga um valor igual ■ sua aposta, ou seja, ele recebe o dobro do que apostou. Nenhuma ficha extra é paga a naturais ou mãos de cinco, e não ■ permitido blefar.

O jogador também não pode dar as cartas ■ fazer papel de banca — no jogo convencional ■ banca pertence ■ quem consegue um natural. Resta, assim, ■ jogador, a difícil tarefa de quebrar ■ banca, o que ocorre quando ele consegue acumular mais de 1 000 fichas. No início da partida, o jogador recebe 100 fichas e, caso perca todas, perde também o jogo.

Carregue as duas seções iniciais do programa e acrescente as linhas seguintes para completar seu jogo.

```
2500 PAUSE 50
```

```
2510 LET DPF=B: LET AF=B: LET 
=C: LET Y=10
2520 FOR J=C TO 
2530 LET Z=O(J): GOSUB 5500
2540 IF VA>10 THEN  LET VA=10
2545 FOR K=C TO 2: LET W(K)=W(K
)+VA: NEXT K
2546 IF VA=C AND AF=B THEN  LET
 W(2)=W(2)+10: LET AF=C
2550 LET X=X+6: NEXT J
2560 IF W(2)=21 THEN  PRINT  PA
PER 2; INK 7;AT 14,18;" NATURAL
! ";AT 15,16;" A BANCA GANHA ":
 LET DPF=C: GOTO 2000
2600 IF PF=C THEN  GOTO 2720
2610 PAUSE 75
2630 IF W(C)>21 THEN  PRINT  PA
PER C; INK 7;AT 21,B;" A BANCA
ESTOURA! ";: GOTO 2740
2635 IF W(2)>21 THEN  LET W(2)=
W(C)
2640 IF W(2)<16 THEN  GOTO 2800
2650 IF FF=C THEN  GOTO 2800
2660 LET PR=(W(2)-8)/13
2670 IF PR<RND THEN  GOTO 2800
2700 IF W(2)>21 THEN  LET W(2)=
W(C)
2710 IF W(2)=21 THEN  PRINT AT
20,B;"  BANCA PAGA SOMENTE NAT
URAIS   E MAOS DE CINCO": GOTO
2000
2720 PRINT  PAPER 2;AT 21,B;" A
 BANCA OBTEVE ";W(2)+C;
2725 IF S(2)>21 THEN  LET S(2)=
S(C)
2730 IF W(2)>=S(2) THEN  PRINT
 PAPER 2;" E GANHOU! ": GOTO 20
00
2740 PRINT  PAPER 2;" VOCE GANH
OU ": LET CP=CP+BET*2: GOTO 200
0
2800 LET Z=C(CC): GOSUB 5500
2805 IF VA>10 THEN  LET VA=10
2810      K=C     2: LET W(K)=W(K
)+VA: NEXT K
2820 IF VA=C AND AF=B THEN  LET
 W(2)=W(2)+10: LET AF=C
2822 IF W(1)<22 AND X=25 THEN
       PAPER 2; INK 7;AT 21,B;"
 MAO DE CINCO! ": GOTO 2000
2825       7000
2830 LET X=X+6: GOTO 2610
```

A linha 2500 provoca um segundo de pausa e, então, programa inicia a jogada da banca. A 2510 coloca zero no sinalizador de naturais e no sinalizador de ases da banca, acerta coordenadas da primeira carta.

DISTRIBUIÇÃO DAS CARTAS

As duas cartas iniciais são "viradas" pelo **FOR...NEXT** entre as linhas 2520 2550; linha 2530 mostra cartas. A 2540 atribui o valor dez às cartas de figuras, antes que a 2545 some seu valor com total da banca (dois totais, se houver um ás). A 2546 dez a **W(2)**, se uma das cartas for um ás, e coloca 1 no indicador de ases. A linha acerta a posição da próxima carta.

A linha 2560 verifica se foi obtido natural, anunciando a vitória da banca, se for o caso. O indicador de naturais da banca passa a valer 1. Se o jogador também tem um natural — linha 2600 —, programa vai rotina que cuida das mensagens: "quem tem o que" e "quem venceu". Após a pausa da linha 2610, linha 2630 verifica se a banca estourou; em caso afirmativo, surge a mensagem A BANCA ESTOUROU. A linha 2635 verifica se maior dos totais — quando há ás — excedeu 21. O programa só considera total. Se banca tiver menos de 16 pontos, a linha vira outra carta.

QUER MAIS CARTAS?

As linhas 2660 2670 decidem se a banca quer outra carta. Não regra fixa para isso. Não vale a pena impor um limite acima do qual a banca não pede mais cartas, pois ela se tornará previsível e o jogador logo saberá como derrotá-la. É necessário introduzir um fator de incerteza (como fazemos aqui), que torne impossível prever quando a banca vai parar de pedir cartas.

Para isso, o programa deverá fazer o computador se comportar de maneira semelhante a um humano. Coloque-se no lugar da banca: tendo um total de 20 pontos, você se sentiria menos inclinado a pedir outra carta do que se tivesse 16. Quando o computador compara PR com um número randômico, introduz-se um fator de incerteza e, ao mesmo tempo, a decisão passa depender do valor da mão da banca.

O RESULTADO FINAL

A linha 2700 troca o valor de **W(2)** pelo de **W(1)**, se **W(2)** contiver mais de 21. Se a banca conseguir 21 pontos, linha 2710 imprime na tela a mensagem A BANCA PAGA SOMENTE NATURAIS E MÃOS DE CINCO. Caso possua uma mão inferior a 21, a linha 2720 escreve A BANCA OBTEVE, seguido de um valor um ponto mais alto que total alcançado pela banca.

A linha 2730 compara total do jogador com o da banca. Quando esta vence, linha 2730 informa o fato; contrário, linha 2740 anuncia vitória do jogador calcula o novo total de fichas.

A parte final do programa — linhas 2800 2830 — cuida da distribuição das cartas da banca.

A linha 2805 atribui dez pontos às cartas de figura. O valor da carta é somado de **W(1)** e **W(2)** na linha 2810. Se houver um ás, à linha 2820 cabe executar as alterações necessárias antes que a linha 2822 verifique se se conseguiu uma mão de cinco. A linha 2825 ajusta monte de cartas, preparando nova distribuição. A linha 2830 calcula a posição onde a carta vai ser desenhada.

Apague o **GOTO** do final da linha 650 e adicione as linhas seguintes:

```
500 GOSUB 4000:GOTO 540
510 CX=CX+32:GOSUB 3500:NC=NC+1
530 IF DL>21 THEN 620
540 DT=DL+10*(DA AND(DL<12)):IF
 DT=21 AND NC=2 THEN GOSUB 5000
:PRINT#1,"A BANCA TEM UM NATURA
L":GOSUB 5500:GOTO 610
550 IF NC=5 THEN GOSUB 5000:PRI
```

```
NT#1,"A BANCA TEM UMA MÃO DE CI
NCO":GOSUB 5500:GOTO 610
560 R=20-DT:IF RND(1)*100<R*R*R
 ■ DT<16 OR PS=1 THEN 510
570 GOSUB 5000:IF P5=1 THEN PRI
NT#1,"VOCE TEM UMA MÃO DE CINCO
" ELSE PRINT#1,"Você tem";PT
580 GOSUB 5500:GOSUB 5000:PRESE
T(8,80):PRINT#1,"A banca paga s
omente";:PRESET(8,92):IF DT<21
THEN PRINT#1,"totais maiores ou
 iguais a";DT+1 ELSE PRINT#1,"n
aturais e mãos de cinco"
590 GOSUB 5500:IF P5=1 THEN 630
600 IF DT<PT THEN 630
610 GOSUB 5000:PRINT#1,"A banca
 venceu":GOSUB 5500:GOTO 690
620 GOSUB 5000:PRINT#1,"A banca
 estourou. Você venceu":GOSUB 5
500:GOTO 640
630 GOSUB 5000:PRINT#1,"Você ve
nceu":GOSUB 5500
640 MN=MN+2*BT:GOSUB 5000:GOSUB
 7000:GOTO 700
650 GOSUB 5000:PRINT#1,"VOCE TE
M UM NATURAL":PF=1:GOSUB 5500
660 GOSUB 4000:IF DL=11 THEN GO
SUB 5000:PRINT#1,"MAS ■ BANCA T
EM UM TAMBEM":GOSUB 5500:GOTO ■
10
670 GOTO 630
690 IF MN<1 THEN GOSUB 5000:PRI
NT#1,"Suas fichas acabaram":GOS
UB 5500:GOTO 790
700 IF PF=1 THEN GOSUB 5000:PRI
NT#1,"Embaralhando as cartas":G
OSUB 5500:GOSUB 1500:GOTO 750
710 NF=N-1:IF NA>N THEN NF=N+51
720 FOR X=NF TO NA+1 STEP -1:Q=
INT■(X-NA)+NA):T=SQ(X):SQ(X)=SQ
(Q):SQ(Q)=T:NEXT:IF NA>N THEN 7
40
730 FOR X=0 TO 9:SQ(X+52)=SQ(X)
:NEXT:GOTO 750
740 FOR X=0 TO 9:SQ(X)=SQ(X+52)
:NEXT
790 GOSUB 5000:PRINT#1,"Quer jo
gar novamente (S/N)?":CLOSE1
800 A$=INKEY$:IF A$<>"N" AND A$
<>"N" THEN 800
810 IF A$="S" THEN RUN
820 COLOR 15,4,4:END
3500 GOSUB 1000:GOSUB 2000:IF ■
M=1 THEN DA=1
3510 IF NM>10 THEN DL=DL+10 ELS
E DL=DL+NM
3520 RETURN
4000 NN=N:N=D1:CX=31:CY=105:GOS
UB 3500
4010 N=D2:CX=63:GOSUB 3500:N=NN
4020 NC=2:RETURN
```

Antes da banca jogar, vêem-se na tela a mão do jogador e as duas cartas da banca fechadas. A banca abre, então, suas cartas — a linha 500 chama a sub-rotina 4000. Esta abre a primeira carta, chamando inicialmente outra sub-rotina, da linha 3500, que cuida do desenho das cartas e calcula os pontos da banca. Na linha 4010 ■ segunda carta é aberta, chamando-se ■ sub-rotina novamente, depois que a posição horizontal da carta foi modificada. O número da carta é atualizado e a sub-rotina termina ■ linha 4020.

A linha 510 muda e acerta a posição horizontal da carta seguinte ■ chama ■ sub-rotina da linha 3500. Assim, a banca recebe uma nova carta. A linha 530 verifica se ■ banca ultrapassou 21 pontos; ■ linha 620 avisa o jogador, ■ for o caso. Se a banca conseguiu um natural, o indicador de naturais da banca passa a valer 1 e o jogador recebe uma mensagem. A linha 550 detecta mãos de cinco.

AS DECISÕES DA BANCA

A parte mais interessante do programa está na linha 560, que faz o computador decidir se quer ou não mais cartas. Seria mais fácil que ■ máquina parasse de pedir cartas quando atingisse um determinado total — 19, por exemplo. Porém, seria igualmente fácil para o jogador derrotar ■ banca, uma vez que soubesse do fato. É necessário, portanto, introduzir um fator de incerteza na decisão do computador, tornando-a imprevisível. Isso não significa que se deva programar uma estratégia suicida — fazendo, por exemplo, com que ■ banca habitualmente peça mais cartas, já tendo 20 pontos.

Criamos, então, uma variável **R**, igual ■ 20, menos os pontos da banca. A seguir, um número qualquer entre 0 ■ 100 é escolhido ■ ■ e comparado com **R*R*R**. Se o número for menor, se ■ banca tiver menos de 16 pontos ou se o jogador conseguir uma mão de cinco, o computador pede mais uma carta. O modo como calculamos ■ faz a chance da banca pedir mais cartas diminuir à medida que seu total de pontos aumenta. Raramente ela pedirá mais cartas já tendo 19 pontos, mas quase sempre o fará quando ainda tiver 16.

Usando o mesmo raciocínio e modificando um pouco a linha 560, podemos, além de variar a dificuldade do jogo, fazer com que a banca jogue baseando-se também nas cartas abertas do jogador, por exemplo.

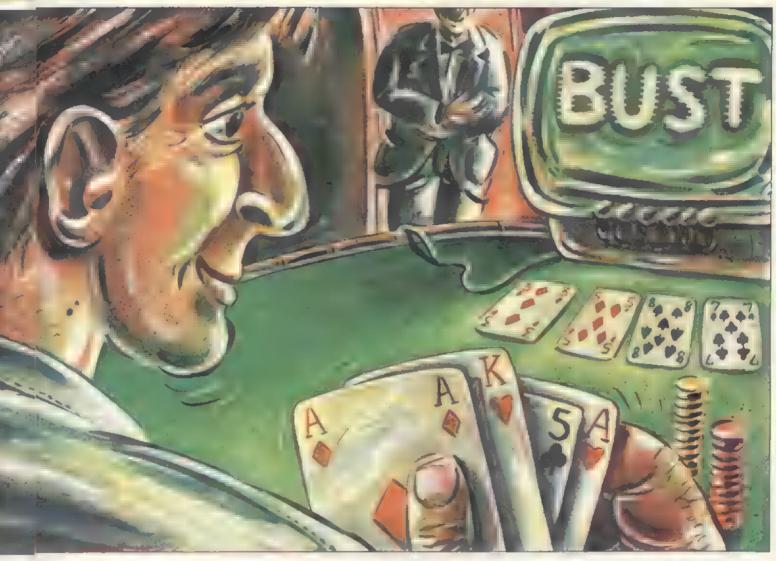
O RESULTADO FINAL

A linha 570 mostra o valor da mão do jogador, apontando a ocorrência de uma mão de cinco, se for o caso. A linha 580 indica as mãos que vencem a banca. Se o jogador tiver uma mão de cinco ■ a banca não, a linha 590 exibe a mensagem VOCÊ VENCEU. De modo semelhante, a linha 600 compara ■ totais do jogador e da banca, informando o resultado através da linha 610 ou da linha 630.

Quando o jogador vence, a linha 640 acrescenta ■ fichas ganhas às que o jogador já tinha. Sempre que alguém consegue um natural, as cartas são embaralhadas. A linha 700 verifica o indicador de naturais e as linhas 710 a 740 embaralham as cartas.

Após ■ anúncio de que o jogador conseguiu um natural, ■ linha 660 verifica se ■ banca obteve o mesmo tipo de mão. Em caso afirmativo, vence a banca e a linha 610 indica o resultado. Se ■ jogador ganhar, caberá à linha 670 informar o fato.

Quando se acabam as fichas do jogador, a linha 690 diz VOCÊ PERDEU TODAS AS FICHAS.

Finalmente, uma pequena rotina (linhas 790 a 820) oferece ao jogador a opção de jogar novamente.

Apague o **GOTO 190** do fim da linha 650 e acrescente as linhas seguintes.

```
500  GOSUB 4000: GOTO 540
510 CX = CX + 54: GOSUB 3500:NC
 = NC + 1
520  FOR K = 1 TO 1700: NEXT K
530  IF DL > 21 THEN 620
540 DT = DL + 10 * (DA AND (DL
< 12)): IF DT = 21 AND NC = 2 T
HEN  TEXT : HOME : PRINT "A BAN
CA TEM UM NATURAL":PF = 1: GOTO
 610
550  IF NC = 5 THEN  TEXT : HOM
E : PRINT "A BANCA TEM UMA MAO
DE CINCO": GOTO 610
560 R = 20 - DT: IF  RND (1) *
100 < R * R * R OR DT < 16 OR P
5 = 1 THEN 510
570  TEXT : HOME : IF P5 = 1 TH
EN  PRINT "VOCE TEM UMA MAO DE
CINCO": GOTO 580
575  PRINT : PRINT "VOCE TEM ";
PT
580  PRINT : PRINT "A BANCA PAG
A ";: IF DT < 21 THEN  PRINT "V
ALORES MAIORES OU IGUAIS A ";DT
 + 1: GOTO 590
585  PRINT "NATURAIS E MAOS DE
CINCO APENAS"
590  FOR K = 1 TO 500: NEXT : I
F P5 = 1 THEN 630
600  IF DT < PT THEN 630
610  PRINT : PRINT "A BANCA GAN
HOU": GOTO 690
620  TEXT : HOME : PRINT "A BAN
CA ESTOUROU. VOCE GANHOU": GOTO
 640
630  PRINT : PRINT "VOCE GANHOU
"
640 MN = MN + 2 * BT: GOTO 700
660  GOSUB 4000: HOME : TEXT :
IF DL = 11 THEN  PRINT "MAS A B
ANCA TEM A MESMA COISA": GOTO 6
10
670  GOTO 630
690  PRINT : IF MN < 1 THEN  PR
INT "VOCE PERDEU TODAS AS FICHA
S": GOTO 790
700  IF PF = 1 THEN  PRINT : PR
INT "EMBARALHANDO AS CARTAS": G
OSUB 1500: GOTO 750
710 NF = N - 1: IF NA > N THEN
NF = N + 51
720  FOR X = NF TO NA + 1 STEP
 - 1:Q =  INT ( RND (1) * (X -
NA)) + NA:T = SQ(X):SQ(X) = SQ(
Q):SQ(Q) = T: NEXT : IF NA > N
THEN 740
730  FOR X = 0 TO 9:SQ(X + 52)
= SQ(X): NEXT : GOTO 750
740  FOR X = 0 TO 9:SQ(X) = SQ(
X + 52): NEXT
790  PRINT : PRINT "QUER JOGAR
NOVAMENTE (S/N) ?"
800  GET A$: IF A$ <  > "S" AND
 A$ <  > "N" THEN 800
810  IF A$ = "S" THEN 150
820  END
3500  POKE  - 16299,0: POKE  -
16304,0: GOSUB 1000: GOSUB 2000
: IF NM = 1 THEN DA = 1
3510  IF NM > 10 THEN DL = DL +
 10: GOTO 3520
3515 DL = DL + NM
3520  RETURN
4000 NN = N:N = D1:CX = 6:CY =
100: GOSUB 3500
```

```
4010 N = D2:CX = 60: GOSUB 3500
:N = NN
4020  FOR ■ = 1 TO 3500: NEXT :
NC = 2: RETURN
```

Como o programa completo é muito extenso, utilizamos a página gráfica 2. Nunca use a página 1, pois o programa será danificado.

Quando a vez de jogar passa ■ banca, as cartas do jogador e ■ duas cartas fechadas do adversário aparecem na tela. A banca, em primeiro lugar, abre suas cartas — ■ linha 500 chama a sub-rotina 4000, que abre a primeira carta da banca chamando outra sub-rotina, da linha 3500. Esta sub-rotina liga o modo gráfico da página 2, sem apagar seu conteúdo, desenha a carta e calcula ■ total de pontos. A linha 4010 abre ■ segunda carta, que chama a ■ sub-rotina após ter modificado a posição horizontal da carta. O número da carta é atualizado e a sub-rotina termina na linha 4020.

A linha 510 calcula a posição horizontal da nova carta e volta a chamar ■ sub-rotina 3500. Assim, a banca recebe mais uma carta. A linha 520 provoca ■ uma pausa antes que a linha 530 verifique se a banca não ultrapassou os 21 pontos — a linha 630 informa o fato, se necessário. A linha 540 verifica se a banca obteve um natural; em caso afirmativo, coloca 1 no indicador de naturais e exibe uma mensagem na tela. A linha 550 detecta uma mão de cinco.

AS DECISÕES DA BANCA

A parte mais interessante do programa está ■ linha 560, que faz o computador decidir se quer ou não mais cartas. Seria mais fácil programar ■ máquina de modo que parasse de pedir cartas sempre que atingisse um determinado valor — 19, por exemplo. Mas seria igualmente fácil para o jogador derrotá-la, uma vez que tivesse conhecimento dessa estratégia inflexível. É necessário introduzir no programa ■ elemento de acaso, que torne as decisões da banca imprevisíveis. Não devemos, porém, programar o computador com uma estratégia suicida — que faça a banca pedir sempre mais cartas, já tendo 20 pontos, por exemplo.

Criamos, assim, uma variável **R**, igual ■ 20, menos o total de pontos da banca. Um número qualquer entre 0 e 100 é escolhido e comparado com **R*R*R**. Se o número for menor, se a banca tiver menos de 16 pontos ou se ■ jogador conseguir uma mão de cinco, ■ computador pede mais ■ carta. A maneira ■ calculamos ■ faz com que ■ chance da banca pedir cartas diminua à medida que seu total de pontos aumenta. Raramente ela pede mais uma carta já tendo 19 pontos, mas quase sempre o fará quando ainda tiver 16.

O RESULTADO FINAL

A linha 570 mostra o total de pontos do jogador. A linha 575 indica uma mão de cinco. As linhas 580 e 585 informam quais os tipos de mão que a banca está pagando.

Se o jogador tiver uma mão de cinco ■ a banca não, a linha 590, após uma pequena pausa, exibe na tela VOCÊ GANHOU. De modo semelhante, a linha 600 compara os totais do jogador e da banca; o resultado é dado pelas linhas 610 ou 630, dependendo do vencedor.

Quando ■ jogador ganha, a linha 640 acrescenta as fichas obtidas às que ele já possuía. Sempre que alguém consegue um natural, as cartas são embaralhadas. A linha 700 verifica o indicador de naturais e as linhas 710 ■ 740 embaralham as cartas.

A linha 660 entra em ação quando o jogador obtém um natural, verificando se a banca conseguiu a mesma coisa — neste caso, a banca ganha e ■ programa segue ■ linha 610. Caso contrário, a 670 informa a vitória do jogador.

A linha 690 avisa que as fichas do jogador acabaram.

Finalmente, as linhas 790 a 820 oferecem-lhe ■ opção de jogar outra vez.

O programa completo é tão extenso que invade a primeira página — MA — da tela do TK-2000. Assim, para concluir o programa, as partes anteriores devem ser carregadas na página dois. Digite **MP** antes de carregar o programa — incompleto ou depois de terminado. Nunca volte ■ **MA**.

As alterações que os usuários do TK-2000 devem fazer para completar seu jogo são ■ mesmas do Apple, com exceção das seguintes linhas (não se esqueça de apagar ■ **GOTO 190** do final da linha 650):

```
540 DT = DL + 10 * (DA ■ (DL
< 12)): IF DT = 21 AND NC = 2 T
HEN  GOSUB 5000: PRINT "A BANCA
 TEM UM NATURAL":PF = 1: GOSUB
6000: GOTO 610
550  IF NC = ■ THEN  GOSUB 5000
: PRINT "A BANCA TEM UMA MAO DE
 CINCO": GOSUB 6000: GOTO 610
570  GOSUB 5000: IF P5 = 1 ■
 PRINT "VOCE TEM UMA MAO DE CI
NCO": GOSUB 6000: GOTO 580
575  GOSUB 5000: PRINT "VOCE TE
M ";PT: GOSUB 6000
580  GOSUB 5000: PRINT "A BANCA
 PAGA ";: IF DT < 21 THEN  PRIN
T "VALORES MAIORES OU IGUAIS A
";DT + 1: GOSUB 6000: GOTO 590
585  PRINT "NATURAIS E MAOS DE
CINCO APENAS": GOSUB 6000
610  GOSUB 5000: PRINT "A BANCA
 GANHOU": GOSUB 6000: GOTO 690
620  GOSUB 5000: PRINT "A BANCA
 ESTOUROU. VOCE GANHOU": GOSUB
6000: GOTO 640
630  GOSUB 5000: PRINT "VOCE GA
NHOU": GOSUB 6000
660  GOSUB 4000: GOSUB 5000: IF
 DL = 11 THEN  PRINT "MAS A BAN
CA TEM A MESMA COISA": GOSUB ■
00: GOTO 610
690  GOSUB 5000: IF ■ < 1 THEN
  PRINT "VOCE PERDEU TODAS AS F
ICHAS": GOSUB 6000: GOTO 790
700  IF PF = 1 THEN  PRINT "EMB
ARALHANDO AS CARTAS": GOSUB 150
0: GOSUB 6000: GOTO 750
790  PRINT "QUER JOGAR NOVAMENT
E (S/N) ?"
3500  GOSUB 1000: GOSUB 2000: ■
F NM = ■ THEN DA = 1
4000 ■ = N:N = D1:CX = 6:CY =
110: GOSUB 3500
```

Você precisará, portanto, recorrer ■ listagem do Apple para fazer as demais modificações.

As diferenças entre o programa do TK-2000 e do Apple devem-se ■ ausência de uma página exclusiva para textos no primeiro computador. Assim, as mensagens têm que ■ impressas na página gráfica.

As explicações sobre o funcionamento do programa são ■ mesmas, com ■ exceção: ■ linha 3500 não ligamos a tela de alta resolução como no Apple, pois nunca saímos dela.

Antes de acrescentar as linhas seguintes, apague o **GOTO 190** do final da linha 650.

```
500 GOSUB 4000:GOTO 540
510 CX=CX+50:GOSUB 3500:NC=NC+1
520 FOR K=1 TO 1725:NEXT
530 IF DL>21 THEN 620
540 DT=DL+10*(DA AND(DL<12)):IF
 DT=21 AND NC=2 THEN CLS:PRINT"
 A BANCA TEM UM NATURAL":PF=1:G
OTO 610
550 IF NC=5 THEN PRINT" A BANCA
 TEM UMA MAO DE CINCO":GOTO 610
560 R=20-DT:IF RND(100)<R*R*R O
■ DT<16 ■■ P5=1 THEN 510
570 CLS:IF P5=1 THEN PRINT" VOC
E TEM UMA MAO DE CINCO" ELSE PR
INT" VOCE TEM";PT
580 PRINT" A BANCA PAGA";:IF DT
<21 THEN PRINT DT+1 ELSE PRINT
" SOMENTE NATURAIS E MAOS DE
   CINCO"
590 FOR K=1 TO 500:NEXT:IF P5=1
 THEN 630
600 IF DT<PT THEN 630
610 PRINT" A BANCA GANHA":GOTO
690
620 CLS:PRINT" A BANCA ESTOUROU
. VOCE GANHA":GOTO 640
630 PRINT " VOCE GANHOU DA BANC
A"
640 MN=MN+2*BT:GOTO 700
660 GOSUB 4000:IF DL=11 THEN ■■
INT:PRINT" MAS A BANCA OBTEVE O
 MESMO VALOR":GOTO 610
670 GOTO 630
690 PRINT:IF MN<1 THEN PRINT "
VOCE PERDEU TODAS ■■ FICHAS":GO
TO 790
700 IF PF=1 THEN PRINT:PRINT" ■
 BANCA EMBARALHA AS CARTAS
APOS OBTER O NATURAL":GOSUB 150
0:GOTO 750
710 NF=N-1:IF NA>N THEN NF=N+51
720 ■■■ X=NF TO NA+1 STEP-1:Q=R
ND(X-NA)+NA-1:T=SQ(X):SQ(X)=SQ(
Q):SQ(Q)=T:NEXT:IF NA>N THEN 74
0
730 FOR X=0 TO 9:SQ(X+52)=SQ(X)
:NEXT:GOTO 750
740 FOR X=0 TO 9:SQ(X)=SQ(X+52)
:NEXT
790 PRINT:PRINT" QUER RECOMECAR
 (S/N)?"
800 A$=INKEY$:IF A$<>"S" AND A$
<>"N" THEN 800
810 IF A$="S" ■■■■ CLS:GOTO 170
820 ■■■
3500 SCREEN 1,1:GOSUB 1000:GOSU
■ 2000:IF NM=1 THEN DA=1
3510 IF NM>10 ■■■■ DL=DL+10 ELS
EDL=DL+NM
3520 RETURN
4000 NN=N:N=D1:CX=6:CY=108:GOSU
B 3500
4010 N=D2:CX=56:GOSUB 3500:N=NN
4020 FOR K=1 TO 2000:NEXT:NC=2:
■■■■■■
```

Quando ■ vez de jogar passa ■ banca, vêem-se na tela ■ mão do jogador e as duas cartas fechadas de seu adversário eletrônico. A banca, em primeiro lugar, abre suas duas cartas — a linha 500 chama a sub-rotina 4000, que abre a primeira carta chamando outra sub-rotina, na linha 3500. Esta liga a tela de alta resolução, desenha a carta e calcula os pontos. A segunda carta é aberta pela linha 4010, que chama ■ ■■■■■ subrotina após ter modificado a posição horizontal da carta. O número da carta é ajustado ■ a sub-rotina termina na linha 4020.

A linha 510 calcula a posição horizontal da nova carta e chama ■ subrotina da linha 3500. Assim, ■ banca recebe mais uma carta. A linha 520 provoca uma pausa antes que ■ 530 verifique se ■ banca tem mais de 21 pontos — ■ linha 620 informa ■ fato, se necessário. A linha 540 verifica se a banca conseguiu ■■ natural; em caso afirmativo, coloca 1 no indicador de naturais ■ exibe uma mensagem na tela. Mãos de cinco são detectadas pela linha 550.

A parte mais interessante do progra■■ está na linha 560, que faz o computador decidir se quer ou não mais cartas. Seria mais fácil programar ■ máqui■■ de maneira que ela parasse de pedir cartas quando atingisse um determinado valor — 19, por exemplo. Mas seria igualmente fácil para o jogador derrotá-la, uma vez que tivesse conhecimento dessa estratégia de jogo. Por isso, é necessário introduzir no programa um elemento de acaso, que torne as decisões da banca imprevisíveis. Por outro lado, não podemos munir ■ programa de uma estratégia suicida, que faça, por exemplo, com que a banca freqüentemente peça mais cartas, já tendo obtido 20 pontos.

Criamos, assim, uma variável **R**, igual ■ 20, menos o valor da mão da banca. O programa escolhe ao acaso um número entre 1 ■ 100 ■ o compara com **R*R*R**. Se o número for menor, se ■ banca tiver menos de 16 pontos ou se o jogador conseguir uma mão de cinco, ■ banca pede mais uma carta. A maneira como calculamos **R** faz com que a chance da banca pedir cartas diminua à medida que seu total de pontos aumenta. Raramente ela pedirá mais cartas tendo já 19 pontos, mas quase sempre o fará quando tiver apenas 16.

O RESULTADO FINAL

A linha 570 mostra o total de pontos do jogador ou indica uma mão de cinco. A linha 580 informa quais os tipos de mão a banca vai pagar.

Se o jogador obteve uma mão de cinco ■ ■ banca não, a linha 590 exibe na tela a mensagem VOCÊ GANHOU. De modo semelhante, ■ linha 600 compara os totais do jogador e da banca; o resultado é dado pelas linha 610 ou 630, dependendo do vencedor.

Quando o jogador ganha, a linha ■■ acrescenta as fichas obtidas às que ele possuía. Sempre que alguém obtém um natural, as cartas são embaralhadas pelas linhas 710 a 740. A linha 700 verifica o indicador de naturais.

A linha 660 entra em ação quando o jogador consegue um natural, verificando se ■ banca conseguiu mão igual. Se ■ banca vencer, a linha 610 anuncia. Caso contrário, a linha 630 informa a vitória do jogador.

A linha 690 avisa que o jogador perdeu todas as fichas.

Finalmente, as linhas 790 a 820 oferecem-lhe ■ opção de jogar novamente.

ROTINAS DE ORDENAÇÃO

Todo tipo de programa que manipula dados pode se beneficiar com o uso de uma rotina de ordenação. Neste artigo mostramos três dos métodos mais empregados para esse fim.

A ordenação é um dos aspectos fundamentais do processamento de informações. Os computadores são muito rápidos na manipulação de dados ■ ■ ordenação destes é um ponto importantíssimo para tornar a informação acessível à análise ■ correção.

Imagine como seria difícil encontrar as referências a um determinado assunto em um livro, sem o auxílio de ■ índice. Ou procurar um número ■ lista telefônica sem que os nomes estivessem em ordem alfabética. Estamos, nos dois casos, diante de listas de informações, ■ quais, ao contrário de uma lista de compras, por exemplo, os itens são arranjados ou ordenados numa ordem específica: a alfabética. Em outros casos — como na ordenação das notas de uma classe —, os itens estariam arranjados em ordem numérica.

Muitos tipos de programa fazem uso de rotinas de ordenação, inclusive os programas de jogos, que comparam placares. Essas rotinas, porém, são mais comumente encontradas em programas que lidam com grande quantidade de dados, como ■ de controle de arquivos, mala direta etc.

O QUE É ORDENAÇÃO

Ordenar é, simplesmente, colocar determinado conjunto de dados em uma ordem específica — algo como arrumar as cartas de um baralho.

A posição relativa de dois itens quaisquer em geral ■ baseia em uma superioridade numérica ou alfabética de um sobre outro.

Em ordenações numéricas, valores maiores vêm sucessivamente antes ou depois dos menores. Em ordenações alfabéticas, as letras são colocadas, sucessivamente, das mais próximas do início até ■ mais próximas do fim do alfabeto, ou vice-versa.

Pode-se também fazer ordenações em base alfanumérica, ou seja, levando-se em conta letras e números. Mas sempre haverá ■ prioridade de uns sobre outros — as letras quase sempre têm mais importância que os números.

Em ordenação numérica, em geral o número 1 tem prioridade, enquanto a letra A assume o primeiro lugar na ordenação alfabética. Mas para fins especiais isso pode ser invertido.

A ORDENAÇÃO TIPO BOLHA

A mais simples rotina para ordenação é ■ chamada ordenação tipo bolha. Você já viu um exemplo dela no artigo da página 292. Agora, examinaremos ■ processo em maior detalhe. Para acompanhar o que acontece, use cartas de baralho ou, então, recorte e numere alguns pedaços de papel.

Coloque esse grupo de cartas (ou pedaços de papel numerados) sobre uma mesa, com o três mais distante de você, na seguinte ordem:

última	[três]
	[sete]
	[dois]
	[seis]
primeira	[dez]

Em uma ordenação tipo bolha, o primeiro valor — aqui, o dez — é comparado com o seguinte do conjunto, havendo uma troca, se ■ valor deste for menor (o que ocorre no nosso caso). O dez é então comparado com ■ dois, com o sete e com o três, mudando sempre de lugar, porque é ■ maior valor ■ cada comparação dois a dois. Poderíamos dizer, usando uma imagem, que o dez "borbulhou" entre os outros valores, o que explica o nome desta rotina. Pelo mesmo raciocínio, valores baixos "borbulham" no sentido inverso.

Assim, depois da primeira passagem, o arranjo das cartas ficou assim:

última	[dez]
	[três]
	[sete]
	[dois]
primeira	[seis]

O processo reinicia com a nova primeira carta, o seis, que é, então, comparada e trocada com ■ anterior, visto

que tem valor maior. Mas, em seguida, ■ comparada com o sete que, por sua vez, ■ maior. As comparações continuam, então, com este valor, enquanto ■ seis fica ■ segunda posição. O sete ■ comparado com ■ três ■ depois com ■ dez, subindo apenas um posto, já que encontrou ■ valor maior.

A ■ ordem das cartas, após ■ segunda passagem, ■ ■ seguinte:

última	[dez]
	[sete]
	[três]
	[seis]
primeira	[dois]

A primeira carta, ■ dois, ■ comparada com ■ anterior. O seis ganha ■ fica ■ mesma posição. Comparado com ■ três, ■ seis troca de lugar com ele, mas fica agora parado nesta posição, uma vez que é ■ que ■ sete ■ ■ dez. A ordenação terminou, pois não há mais trocas ■ se fazer.

A ordem das cartas é, então:

última	[dez]
	[sete]
	[seis]
	[três]
primeira	[dois]

O computador faria ■ última passagem por essa nova ordem das cartas, para se certificar de que não há mais trocas ■ fazer, e só pararia ■ realmente não houvesse.

Durante ■ execução da rotina um certo número de comparações e trocas foi feito. Essas duas operações estão incluídas em qualquer rotina de ordenação. A diferença entre tais rotinas consiste apenas no número de operações que são executadas e, conseqüentemente, no tempo gasto.

Vejamos agora outros exemplos, usando esta seqüência de números:

início							**fim**
67	35	72	19	47	38	11	96

O primeiro da lista é o 67. Ele ■ comparado ■ trocado com o 35, mas pára aí. O maior valor neste ponto é 72, que ■ comparado e trocado sucessivamente com 19, 47, 38, 11, até que encontra ■ 96; ■ rotina, então, recomeça ■ comparação desde o primeiro número.

A lista abaixo mostra todo o processo. Os itens submetidos a comparação estão entre parênteses.

```
( 67 )( 35 )  72      ■     47      ■     11      ■
   ■   ( 67 )( 72 )   ■     47     38     11      ■
  35     67  ( 72 )( 19 )   47      ■     11     96
  35     67      ■ ( 72 )( 47 )     ■     11     96
   ■     67      ■     ■ ( 72 )( 38 )     11     96
   ■     67     19    47      ■ ( 72 )( 11 )     96
  35     67     19    47      ■     11 ( 72 )( 96 )
( 35 )( 67 )     ■    47      ■     11     72      ■
   ■   ( 67 )( 19 )   47      ■     11     72     96
  35     19  ( 67 )( 47 )    38     11     72     96
  35     19     47 ( 67 )( 38 )     11     72     96
  35      ■     47    38 ( 67 )( 11 )      72     96
   ■      ■     47     ■     11 ( 67 )( 72 )      ■
   ■      ■     47     ■     11      ■ ( 72 )( 96 )
( 35 )( 19 )    47     ■     11      ■      ■      ■
   ■   ( 35 )( 47 )    ■     11     67      ■      ■
  19     35  ( 47 )( 38 )    11     67      ■     96
   ■     35      ■ ( 47 )( 11 )      ■     72      ■
   ■     35      ■    11 ( 47 )( 67 )      72     96
   ■     35      ■    11     47 ( 67 )( 72 )      96
   ■     35      ■    11     47      ■ ( 72 )( 96 )
( 19 )( 35 )    38    11     47     67      ■      ■
   ■   ( 35 )( 38 )   11     47      ■     72      ■
  19      ■  ( 38 )( 11 )    47     67     72     96
  19      ■     11 ( 38 )( 47 )     67     72     96
```

```
   19      35      11      38    ( 47 )( 67 )   72      96
   19      35      11      38      47   ( 67 )( 72 )    96
   19      35      11      38      47     67   ( 72 )( 96 )
 ( 19 )( 35 )      11      38      47     67     72      96
   19   ( 35 )( 11 )       38      47     67     72      96
   19      11   ( 35 )( 38 )       47     67     72      96
   19      11      35   ( 38 )( 47 )      67     72      96
   19      11      35      38    ( 47 )( 67 )   72      96
   19      11      35      38      47   ( 67 )( 72 )    96
   19      11      ■       ■       47     62   ( 72 )( 96 )
 ( 19 )( 11 )      35      38      47     67     72      96
   11   ( 19 )( 35 )       38      47     67     72      96
   11      19   ( 35 )( 38 )       47     67     72      96
   11      19      35   ( 38 )( 47 )      67     72      96
   11      19      35      38    ( 47 )( 67 )   72      96
   11      19      35      38      47   ( 67 )( 72 )    96
   11      19      35      38      47     67   ( 72 )( 96 )
 ( 11 )( 19 )      35      38      47     67     72      96
   11   ( 19 )( 35 )       38      47     67     72      96
   11      19   ( 35 )( 38 )       47     67     72      96
   11      19      35   ( 38 )( 47 )      67     72      96
   11      19      35      38    ( 47 )( 67 )   72      96
   11      19      35      38      47   ( 67 )( 72 )    96
   11      19      35      38      47     67   ( 72 )( 96 )
```

Vejamos agora como se comporta ■ rotina de ordenação tipo bolha na forma de programa. Ela está colocada como a sub-rotina 1000.

Grave ■ programa para que possa adicionar outras rotinas mais tarde.

```
 10 POKE 23658,8: LET T=0:
 INPUT "NUMERO DE ITENS ";AA:
 IF AA<2 THEN  GOTO 10
 15 DIM A(AA)
 20 PRINT '''"TABELA DESORDENA
 DA": PRINT
 30 FOR Z=1 TO AA
 40 LET A(Z)=INT (RND*100)+1
 50 PRINT TAB T;A(Z);: LET T=T
 +4: IF T>30 THEN  LET T=0
 60 NEXT Z
 70 PRINT : PRINT : PRINT "PRE
 SSIONE '0' PARA ORDENAR"
 80 LET K$=INKEY$: IF K$<>"0"
 THEN  GOTO 80
 90 GOSUB 1000
100 PRINT : PRINT "TABELA ORDE
NADA": PRINT
110 LET T=0: FOR Z=1 TO ■
120 PRINT TAB T;A(Z);: LET T=T
+4: IF T>30 THEN  LET T=0
130 NEXT Z
140 GOTO 10
999 REM ORDENACAO BOLHA(BUBBLE
SORT)
1000 FOR Z=1 TO AA-1
1010 LET ZZ=0
1020 FOR Y=1 TO AA-Z
1030 IF A(Y+1)<A(Y) THEN  LET ■
=A(Y): LET A(Y)=A(Y+1): LET A(Y
+1)=X: LET ZZ=1
1040 NEXT ■
1050 IF ZZ=0 THEN  RETURN
1060 NEXT Z: RETURN
```

```
10 PRINT:PRINT:INPUT"NUMERO DE
ITENS";AA:IF AA<2 THEN 10
15 DIM A(AA)
20 PRINT:PRINT"TABELA DESORDENA
DA":PRINT
30 FOR Z=1 TO AA
40 A(Z)=INT(RND(1)*100)+1
50 PRINT A(Z),
60 NEXT Z
70 PRINT:PRINT:PRINT"PRESSIONE
'0' PARA ORDENAR"
80 GET K$:IF K$<>"0" THEN 80
90 GOSUB 1000
100 PRINT:PRINT:PRINT"TABELA ■
DENADA"
110 PRINT:FOR Z=1 TO AA
120 PRINT A(Z),
130 NEXT Z
140 ■
999 ■ ORDENACAO BOLHA(BUBBLE
SORT)
1000 FOR Z=1 TO AA-1
1010 ZZ=0
1020 FOR Y=1 TO AA-Z
1030 IF A(Y+1)<A(Y) THEN X=A(Y)
:A(Y)=A(Y+1):A(Y+1)=X:ZZ=1
1040 ■ Y
1050 IF ZZ=0 THEN RETURN
1060 NEXT Z:RETURN
```

O programa acima roda apenas no Apple ■ no TK-2000. Faça as seguintes alterações para adaptar ■ programa à sua máquina:

```
40 A(Z)=RND(100)
50 PRINT A(Z);
80 K$=INKEY$:IF K$<>"0" THEN 80
120 PRINT A(Z);
```

```
5 R=RND(-TIME)
80 K$=INKEY$:IF K$<>"0" THEN 80
1030 IF A(Y+1)<A(Y) THEN SWAP A
(Y),A(Y+1):ZZ=1
```

Execute ■ programa. Inicialmente, ele pedirá a quantidade de itens que você quer ordenar, criando um conjunto de números pseudo-randômicos baseado na sua resposta. Estes são mostrados na tela sob o título "TABELA DESORDENADA". Uma mensagem pede que você tecle '0' para iniciar ■ ordenação.

A rotina começa a funcionar, concluindo a ordenação em cerca de um segundo, ■ os itens forem poucos. Em seguida, o conjunto de números é exibido na tela. No artigo da página 292 você encontrará mais detalhes sobre esta rotina e verá também o algoritmo em forma de diagrama.

Para que você possa comparar ■ diversas rotinas, ■ interessante que cronometre o tempo gasto em cada uma delas. Incorpore ■ rotina seguinte ao programa, ■ fim de que ■ computador faça isso para você. Os usuários do Apple II ou compatível precisarão utilizar o seu relógio...

```
90 TIMER=0:GOSUB 1000:PRINT:PRI
NT" TEMPO: ";TIMER/50;"SEGUNDOS
"
```

```
90 POKE 16920,0:POKE 16919,0:GO
SUB 1000:PRINT"Tempo:";PEEK(169
20);"m";PEEK(16919);"s"
```

```
90 POKE 23672,0: POKE 23673,0
: GOSUB 1000: PRINT : PRINT (
PEEK 23672+256*PEEK 23673)/50
;" SEGUNDOS"
```

```
■ TIME=0:GOSUB1000:PRINT:PRINT
"Tempo: ";TIME/60;" segundos"
```

Para iniciar ■ cronometragem e ■ ordenação, pressione '0' quando ■ mensagem aparecer. O tempo gasto pela rotina para ordenar os dados é exibido e o programa recomeça automaticamente.

Provavelmente você gastaria muito mais tempo do que ■ máquina para executar a mesma tarefa. Porém, pelos padrões computacionais, o tempo gasto pela ordenação tipo bolha é extremamente longo. Por essa razão, não se costuma usá-la em sua forma original ■ programas de manipulação de dados, ■ não ser para listas muito pequenas.

Mas veja que, surpreendentemente, ela se mostra mais veloz que todas as outras quando ■ trata de reordenar uma lista à qual se adicionou um item. Em um programa que armazena endereços comerciais, por exemplo, podemos ordenar as novas entradas separadamente ■ depois incorporá-las à lista principal. Outra alternativa ■ reordenar a lista toda ■ cada nova entrada.

Nessas circunstâncias, a rotina tipo bolha não faz nada mais que comparar o novo dado com os outros, até que sua posição correta seja encontrada. A reordenação de uma lista equivale, assim, à comparação de dois itens. E isso é feito rapidamente.

A ROTINA DE SHELL

As duas versões mais utilizadas da rotina tipo bolha são as rotinas de Shell e de Shell-Metzner. Ambas devem ser consideradas quando o número de itens a serem ordenados ultrapassa ■ casa dos dez.

A rotina de Shell emprega uma técnica de busca binária que funciona dividindo sucessivamente a lista original ao meio, até que ■ posição do item em questão seja encontrada. A cada divi-

são, um novo par de valores é comparado. A rotina termina quando faz uma passagem completa sem trocas.

Uma rotina de Shell tem, tipicamente, a forma mostrada aqui. Adicione-a ao seu programa e mude o número do **GOSUB** da linha 90 para testá-la.

```
1999 REM ORDENACAO SHELL
2000 LET Z=AA
2010 IF Z<=1 THEN  RETURN
2020 LET Z=INT (Z/2): LET Y=AA-
Z
2030 LET ZZ=0
2040 FOR X=1 TO Y
2050 LET XX=X+Z
2060 IF A(X)>A(XX) THEN  LET YY
=A(X): LET A(X)=A(XX): LET A(XX
)=YY: LET ZZ=1
2070 NEXT X
2080 IF ZZ>0 THEN  GOTO 2030
2090 GOTO 2010
```

```
1999 REM ORDENACAO SHELL
2000 Z=AA
2010 IF Z<=1 THEN RETURN
2020 Z=INT(Z/2):Y=AA-Z
2030 ZZ=0
2040 FOR X=1 TO Y
2050 XX=X+Z
2060 IF A(X)>A(XX) THEN YY=A(X)
:A(X)=A(XX):A(XX)=YY:ZZ=1
2070 NEXT X
2080 IF ZZ>0 THEN 2030
2090 GOTO 2010
```

PARA UMA ORDENAÇÃO MAIS RÁPIDA

A velocidade de qualquer rotina de ordenação aumenta muito quando a informação é oferecida já semi-ordenada. Isso é importante sobretudo no caso da ordenação tipo bolha.

Se os itens vão ser ordenados cronologicamente, por exemplo, devemos armazenar as datas na forma ANO/MÊS/DIA, em vez da mais usual DIA/MÊS/ANO. Dessa maneira, aplica-se a ordenação inicialmente ao grupo completo e, depois, aos subgrupos.

O processo seria muito mais demorado se ordenássemos inicialmente os dias, para depois reordenar alguns dados em função dos meses, repetindo finalmente todo o trabalho em função dos anos.

Uma sub-rotina adicional poderia ser usada para inverter a seqüência usual de entrada da data para uso da rotina de ordenação.

Se o seu micro é da linha MSX, mude a linha 2060 para:

```
2060 IF A(X)>A(XX) THEN SWAP A(
X),A(XX):ZZ=1
```

Para melhor compreender o funcionamento da rotina, vamos examinar em detalhe um exemplo. Suponhamos que devemos ordenar uma lista de números apresentada nesta ordem:

67 35 72 19 47 38 11 96

A rotina divide a lista em duas outras e compara o primeiro valor de cada uma delas. Se o primeiro valor da primeira lista é maior, ocorre uma troca. Neste exemplo, os valores 67 e 47 são comparados e trocados.

47 35 72 19 : 67 38 11 96

Após a troca, o par de valores seguinte é comparado — 35 e 38. Nesse caso, não há troca. A rotina compara, então, 72 e 11, trocando-os e, por fim, 19 e 96, obviamente sem troca. A lista fica assim:

47 35 11 19 : 67 38 72 96

Neste ponto, cada metade é novamente dividida:

47 35 : 11 19 : 67 38 : 72 96

A rotina compara e troca o primeiro valor, 47, com o terceiro, 11. Depois, compara e troca o segundo, 35, com o quarto, 19. O 35 assume agora a quarta posição. O valor no terceiro posto, 47, que já foi trocado uma vez, é comparado com 67. Desta vez não há troca. O quarto valor, 35, é comparado com 38 e permanece onde está. O quinto valor, 67, é comparado com o sétimo, 72, e fica na mesma posição. Por fim, 38 e 96 são comparados e não trocam de lugar.

Veja abaixo a seqüência. Como no exemplo anterior, os valores comparados estão entre parênteses. Verifique se, na linha seguinte, alguma troca foi efetuada:

(47) 35 : (11) 19 : 67 38 : 72 96
11 (35) : 47 (19) : 67 38 : 72 96
11 19 : (47) 35 : (67) 38 : 72 96
11 19 : 47 (35) : 67 (38) : 72 96
11 19 : 47 35 : (67) 38 : (72) 96
11 19 : 47 35 : 67 (38) : 72 (96)
11 19 : 47 35 : 67 38 : 72 96

A lista é, então, dividida da seguinte maneira:

11 : 19 : 47 : 35 : 67 : 38 : 72 : 96

Os valores são novamente comparados dois a dois. Como você pode observar, a rotina compara somente valores adjacentes: o primeiro com o segundo, o segundo com o terceiro, o terceiro com o quarto e assim por diante, até o fim da lista:

(11) : (19) : 47 : 35 : 67 : 38 : 72 :96
11 : (19) :(47): 35 : 67 : 38 : 72 :96
11 : 19 :(47):(35): 67 : 38 : 72 :96
11 : 19 : 35 :(47):(67): 38 : 72 :96
11 : 19 : 35 : 47 :(67):(38): 72 :96
11 : 19 : 35 : 47 : 38 :(67):(72):96

A rotina faz o mesmo até os últimos valores, que já estão em ordem, e volta ao início:

(11) : (19) : 35 : 47 : 38 : 67 : 72 :96
11 : (19) :(35): 47 : 38 : 67 : 72 :96
11 : 19 :(35):(47): 38 : 67 : 72 :96

```
11 : 19 : 35 :(47):(38): 67 : 72 :96
11 : 19 : 35 : 38 :(47):(67): 72 :96
```

Embora a lista já esteja ordenada, o computador executa mais uma passagem completa. Cada novo valor pode agora ser comparado com essa lista como o faz a rotina tipo bolha — isto é, estabelecendo comparações aos pares.

A ROTINA DE SHELL-METZNER

A rotina de ordenação de Shell-Metzner, derivada da rotina de Shell, é ainda mais rápida e utiliza o mesmo princípio de busca binária.

```
2999 REM SHELL-MELTZNER SORT
3000 LET Z=AA
3010 LET Z=INT (Z/2)
3020 IF Z=0 THEN  RETURN
3030 LET Y=AA-Z: LET ZZ=1
3040 LET X=ZZ
3050 LET XX=X+Z
3060 IF A(X)<=A(XX) THEN  GOTO
 3090
3070 LET W=A(X): LET A(X)=A(XX)
: LET A(XX)=W: LET X=X-Z
3080 IF X>=1 THEN  GOTO 3050
3090 LET ZZ=ZZ+1
3100 IF ZZ<=Y THEN  GOTO 3040
3110 GOTO 3010
```

```
2999 REM ORDENACAO 'SHELL-METZN
ER'
3000 LET Z=AA
3010 Z=INT(Z/2)
3020 IF Z=0 THEN RETURN
3030 Y=AA-Z:ZZ=1
3040 X=ZZ
3050 XX=X+Z
3060 IF A(X)<=A(XX) THEN 3090
3070 W=A(X):A(X)=A(XX):A(XX)=W:
X=X-Z
3080 IF X>=1 THEN 3050
3090 ZZ=ZZ+1
3100 IF ZZ<=Y THEN 3040
3110 GOTO 3010
```

Se você trabalha com um MSX, beneficie-se de seu poderoso BASIC, trocando a linha 3070 para:

```
3070 SWAP A(X),A(XX):X=X-Z
```

Adicione esta rotina ao seu programa de demonstração e para experimentá-lo será preciso modificar a referência de linha da linha 90.

Como você vai notar, ela é muito mais rápida que as outras rotinas, o que fica evidente especialmente se a compararmos com a rotina tipo bolha para mais de cinqüenta números.

DRAGÃO ANIMADO

Aventura que se preze, com reis, rainhas e castelos, não dispensa [illegible] participação de um temível dragão.

Não é difícil incluir esse personagem em seus jogos. Para criá-lo e dar-lhe animação — fazendo com que cuspa fogo —, você pode utilizar as técnicas já vistas nos programas do sapo e do tanque, apresentados [illegible] artigo publicado à página 341.

Para montar o "quadriculado" na memória do computador, usaremos o programa dado no artigo anterior. O Apple, o TK-2000 e também o MSX não precisam desse tipo de programa. A listagem para o Spectrum e o TRS-Color encontra-se naquele artigo.

Depois de carregar — ou digitar — o programa criador de quadriculado, use **RUN** para fazê-lo funcionar. Quando [illegible] programa acabar de rodar, digite **NEW** [illegible] **<ENTER>**. (Nada disso se aplica ao MSX, ao Apple e ao TK-2000.) Em seguida, digite [illegible] programa adequado para seu computador.

Os usuários do Apple, do TK-2000 e do MSX precisarão carregar — ou digitar — o programa do tanque, pois as linhas apresentadas neste artigo são modificações [illegible] serem feitas naquele programa, [illegible] não funcionarão sozinhas.

Em todos os computadores, exceto no MSX, [illegible] as teclas **P**, **L**, [illegible] e **Z** para mover [illegible] dragão. No MSX devem ser utilizadas as teclas do cursor. A barra de espaço faz o dragão cuspir fogo.

```
10 FOR n=USR "a" TO USR "t"+7
 : [illegible] a: POKE n,a: NEXT n
20 LET print=32400: LET b=
32402: IF PEEK 23733=255 THEN
LET print=65200: LET b=65202
90 BORDER 7: PAPER 7: [illegible] 4:
CLS : [illegible] AT 8,15;: [illegible]
USR print
100 LET y=8: LET x=15: LET y1=
8: LET x1=15: LET z=1
```

Vimos, [illegible] artigo anterior, como desenhar [illegible] dar movimento [illegible] um tanque [illegible] guerra e [illegible] um sapo. Com [illegible] técnicas e programas já apresentados, desenhe também um dragão que cospe fogo.

- COMO COLOCAR UMA NOVA FIGURA NO QUADRICULADO
- MOVIMENTOS NA TELA
- FAÇA O DRAGÃO CUSPIR FOGO

```
110 LET a$-INKEY$: IF a$-""
THEN  GOTO 110
115 IF a$-" " THEN  GOTO 200
120 IF a$-"z" AND x>1 THEN
LET x1-x-1: LET z-1
130 IF a$-"x" [illegible] x<28 THEN
LET x1-x+1: LET z-2
140 IF a$-"p" [illegible] y>0 [illegible]
LET y1-y-1
150 IF a$-"1" [illegible] y<19 THEN
LET y1-y+1
160 PRINT AT y.x;: POKE b,0:
RAND USR print
170 LET x-x1: LET y-y1
180 PRINT AT y.x;: POKE b,z:
RAND USR print
190 GOTO 110
200 IF z-2 THEN  GOTO 300
220 PRINT  INK 6;AT y.x-1;CHR$
162
230 PAUSE 1
240 PRINT AT y.x-1;" "
260 GOTO 110
300 PRINT  INK 6;AT y.x+3;CHR$
163
310 PAUSE 1
320 PRINT AT y.x+3;" "
330 GOTO 110
1000 DATA 6,214,249,63,240,0,3,
15,96,64,192,224,224,192,196,20
4,0,0,0,0,0,0,0,0
1010 DATA 63,125,107,245,249,12
7,127,51,244,196,194,255,128,19
2,224,240,0,0,0,2,6,14,6,10
1020 DATA 55,23,7,3,1,0,4,7,254
,255,239,239,224,192,128,128,56
,240,192,0,0,0,0,0
1030 DATA 0,0,0,0,0,0,0,0,6,2,3
,7,7,3,35,51,96,107,159,252,15,
0,192,240
1040 DATA 0,0,0,64,96,112,96,80
,47,33,65,255,1,3,7,15,252,190,
182,175,159,254,254,204
1050 DATA 28,15,3,0,0,0,0,0,127
,255,247,247,7,3,1,1,236,232,22
4,192,128,0,32,224
1060 DATA 8,68,50,139,50,68,8,0
,16,34,76,145,76,34,16,0
5000 [illegible] i-1 TO 5
5010 [illegible] "MC0301"
5020 NEXT i
```

```
20 PCLEAR 5
```

```
30 CLEAR 200,32000
40 FOR I=32300 TO 32587
50 [illegible]
60 POKE I,N
70 NEXT
80 DIM A(2),B(2),C(2)
90 PMODE 4,1
100 PCLS
110 FOR I=1536 TO 1760 STEP 32
120 READ [illegible]
130 POKE I,N
140 NEXT
150 GET (0,0)-(7,7),A
160 FOR I=1536 TO 1760 STEP 32
170 READ [illegible]
180 POKE I,N
190 NEXT
200 GET (0,0)-(7,7),B
210 PCLS
220 SCREEN 1,0
230 T=1
240 DP=3500
250 POKE 32700,INT(DP/256)
260 POKE 32701,DP-256*INT(DP/25
6)
270 POKE 32250,T+DT*2
280 EXEC 32000
290 LP=DP
300 IF PEEK(338)=251 THEN DP=DP
-32:GOTO 360
310 IF PEEK(342)=253 THEN DP=DP
+32:GOTO 360
320 IF PEEK(340)=247 THEN DP=DP
-1:T=T+1:DT=1:GOTO 360
330 IF PEEK(338)=247 THEN DP=DP
+1:T=T+1:DT=0:GOTO 360
340 IF INKEY$=" " AND T=2 GOSUB
 410
350 GOTO 300
360 IF DP<1536 OR DP>6941 THEN
DP=LP
370 IF T>2 THEN T=1
380 POKE 32250,0
390 EXEC 32000
400 GOTO 250
410 IF DT=1 GOTO 470
420 YP=INT((DP-1536)/32)
430 XP=8*(DP-1533-YP*32)
440 IF XP>255 THEN 530
450 PUT (XP,YP)-(XP+7,YP+7),A
460 GOTO 510
470 YP=INT((DP-1536)/32)
480 XP=8*(DP-1537-YP*32)
490 IF XP<0 THEN 530
500 PUT (XP,YP)-(XP+7,YP+7),B
510 FOR I=1 TO 200:NEXT
520 PUT(XP,YP)-(XP+7,YP+7),C
530 RETURN
540 DATA 0,0,0,0,0,0,0,0,6,2,3,
7,7,3,35,51,96,107,159,252,15,0
,192,240
550 DATA 0,0,0,0. 64, 96, 112, [illegible]
6,47,33,65,255,3,7,
15, 31, 248,188 182,
175,159,255,254,254
560 DATA
80,28,15,7,1,0,0,0,
63,255,255,255,255,
15,2,3,252,248,252,
252, 248, 48,82,222
570 DATA 0,0,0,0,0,0,0,0,6,2,
3,7,7,3,33,0,96,107,159,252,15,
192,240
580 DATA 0,0,0,64,96,112,96,80,
49,47,33,47,49,99,7,15,252,190,
182,175,159,254,254,156
590 DATA 28,15,3,0,0,0,0,0,127,
255,239,239,15,6,2,3,220,216,19
2,128,0,0,64,192
[illegible] DATA 6,214,249,63,240,0,3,1
5,96,64,192,224,224,192,196,204
,0,0,0,0,0,0,0,0
610 DATA 31,61,109,117,249,255,
127,127,244,196,194,255,192,224
,240,248,0,0,0,0,2,6,14,6
620 DATA 63,31,63,63,31,12,74,1
23,252,255,255,255,255,240,64,1
92,10,56,240,224,128,0,0,0
630 DATA 0,6,214,249,63,240,3,1
5,0,96,64,192,224,224,192,132,0
,0,0,0,0,0,0,0
640 DATA 63,125,109,245,249,127
,127,57,140,244,132,244,140,198
,224,240,0,0,0,2,6,14,6,10
650 DATA 59,27,3,1,0,0,2,3,254,
255,247,247,240,96,64,192,56,24
0,192,0,0,0,0,0
660 DATA 0,16,34,76,145,76,34,1
6
670 DATA 0,8,68,50,137,50,68,8
```

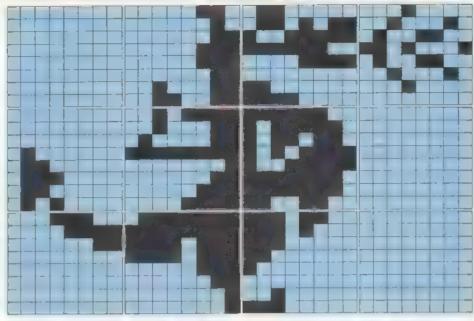
O dragão do TRS-Color é formado por blocos gráficos de oito por oito pontos.

As linhas a seguir são modificações que devem ser feitas no programa que move o tanque de guerra.

```
5 CLEAR 1000:COLOR 12,15,12
20 FOR I=1 TO 32*4
25 READ A:A$=A$+CHR$(A)
30 NEXT I
35 FOR I=1 TO 32*4
40 READ B:B$=B$+CHR$(B)
45 NEXT I
50 SPRITE$(0)=LEFT$(A$,32)
55 FOR I=1 TO 3
60 SPRITE$(I)=MID$(A$,I*32+1,32
)
65 NEXT I
70 SPRITE$(4)=LEFT$(B$,32)
75 FOR I=5 TO 7
80 SPRITE$(I)=MID$(B$,(I-4)*32+
1,32)
85 NEXT I
110 X=100:Y=90:F=1
111 PUTSPRITE0,(X+16,Y-8),12,0
112 PUTSPRITE1,(X+16,Y+8),12,1
113 PUTSPRITE2,(X,Y+8),12,2
500 PUTSPRITE0,(X+16,Y-8),12,0
510 PUTSPRITE1,(X+16,Y+8),12,1
515 PUTSPRITE2,(X,Y+8),12,2
516 PUTSPRITE3,(X+32,209),12,3
600 PUTSPRITE0,(X+16,Y-8),12,4
610 PUTSPRITE1,(X+16,Y+8),12,5
615 PUTSPRITE2,(X+32,Y+8),12,6
616 PUTSPRITE3,(X,209),12,7
700 PUTSPRITE0,(X+16,Y-8),12,0
710 PUTSPRITE1,(X+16,Y+8),12,1
715 PUTSPRITE2,(X,Y+8),12,2
716 PUTSPRITE3,(X+32,Y),6,3
800 PUTSPRITE0,(X+16,Y-8),12,4
810 PUTSPRITE1,(X+16,Y+8),12,5
815 PUTSPRITE2,(X+32,Y+8),12,6
816 PUTSPRITE3,(X,Y),6,7
5000 DATA 0,0,0,0,0,0,0,0,6,2,3
,7,7,3,35,51,0,0,0,0,0,0,0,0,96
,107,159,252,15,0,192,240
5010 DATA 47,35,67,255,1,3,7,15
,127,255,247,247,7,3,1,1,252,19
0,182,175,159,254,254,204,232,2
24,224,192,128,0,32,224
5020 DATA 0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,64,96,112,96,8
0,28,15,3,0,0,0,0,0
5030 DATA 16,34,76,209,76,34,16
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0
```

```
5040 DATA 0,0,0,0,0,0,0,0,6,214
,249,63,240,0,1,7,0,0,0,0,0,0,0
,0,96,64,192,224,224,192,196,20
█
5050 DATA 63,125,109,245,249,12
7,127,51,23,7,3,1,0,0,4,7,244,1
96,194,255,128,192,224,240,254,
255,239,239,224,128,128,192
5060 DATA 0,0,0,2,6,14,6,10,56,
240,192,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0
5070 DATA 0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,8,68,50,139,50,68,8,
0,0,0,0,0,0,0,0,0
```

Carregue o programa que move um tanque na tela usando **DRAW**; faça nele ■ seguintes modificações:

```
30   FOR █ = E TO E + 41 + 18 *
32
110   IF K$ = "Z" ███ X > 16 THE
N █ = X - 3:F = 0: GOTO 140
170   DRAW 1 AT LX,LY + 8
180   DRAW █ AT LX,LY + 16
190   DRAW 3 AT LX + 8,LY
200   DRAW █ AT LX + 8,LY + 8
205   DRAW █ AT LX + 8,LY + 16
210   DRAW 6 AT LX + 16,LY
220   DRAW 7 AT LX + 16,LY + 8
225   DRAW 8 AT LX + 16,LY + 16
260   DRAW 11 AT LX,LY
270   DRAW 12 AT LX,LY + █
275   DRAW 13 AT LX,LY + 16
280   DRAW 14 AT LX + 8,LY
290   DRAW 15 AT LX + 8,LY + 8
295   DRAW 16 AT LX + 8,LY + 16
300   DRAW 17 AT LX + 16,LY + 8
310   DRAW 18 AT LX + 16,LY + 16
340   DRAW 1 AT X,Y + █
350   DRAW █ AT X,Y + 16
360   DRAW █ AT X + 8,Y
370   DRAW █ AT X + 8,Y + 8
375   DRAW 5 AT X + 8,Y + 16
380   DRAW █ AT █ + 16,Y
390   DRAW 7 AT █ + 16,Y + 8
395   DRAW 8 AT X + 16,Y + 16
420   DRAW 11 AT X,Y
430   DRAW 12 AT X,Y + █
435   DRAW 13 AT X,Y + 16
440   DRAW 14 AT X + 8,Y
450   DRAW 15 AT X + 8,Y + 8
455   DRAW 16 AT X + 8,Y + 16
460   DRAW 17 AT █ + 16,Y + █
470   DRAW 18 AT █ + 16,Y + 16
510   DRAW 9 AT X + 30,Y
540   DRAW 9 AT █ + 30,Y
570   DRAW 10 AT X - 12,Y
600   DRAW 10 AT █ - 12,Y
2000  DATA  20 ,0 ,42 ,0 ,74 ,0
 ,106 ,0 ,138 ,0 ,170 ,0 ,202 ,
0 ,234 ,█ ,10 ,1 ,42 ,1 ,74 ,1
,106 ,1 ,138 ,1 ,170 ,1 ,202 ,1
 ,234 ,█ ,10 ,2 ,42 ,2 ,74 ,2 ,
106 ,2 ,138 ,2
2010  DATA   0 ,72 ,73 ,73 ,218
 ,219 ,219 ,74 ,73 ,73 ,218 ,21
9 ,27 ,87 ,109 ,73 ,209 ,219 ,5
9 ,191 ,41 ,77 ,73 ,218 ,219 ,3
1 ,23 ,0 ,0 ,0 ,0 ,0
```

■ Spectrum, ■ MSX, o Apple e o ███ utilizam este dragão como modelo.

```
2020  DATA   0 ,72 ,41 ,109 ,20
9 ,63 ,255 ,155 ,73 ,73 ,173 ,2
19 ,219 ,155 ,73 ,73 ,137 ,219
,219 ,155 ,73 ,73 ,137 ,219 ,21
█ ,155 ,0 ,0 ,0 ,0 ,0 ,0
2030  DATA   0 ,72 ,73 ,109 ,21
█ ,223 ,219 ,74 ,73 ,41 ,213 ,6
3 ,223 ,155 ,73 ,9 ,45 ,213 ,25
█ ,219 ,83 ,105 ,9 ,173 ,59 ,22
3 ,255 ,2 ,0 ,0 ,0 ,0
2040  DATA   0 ,72 ,13 ,45 ,173
 ,59 ,223 ,251 ,10 ,77 ,9 ,173
,59 ,63 ,63 ,63 ,78 ,73 ,9 ,213
 ,255 ,219 ,83 ,73 ,41 ,173 ,59
 ,63 ,223 ,19 ,0 ,0
2050  DATA   0 ,8 ,45 ,45 ,45 ,
213 ,63 ,63 ,63 ,55 ,45 ,109 ,4
█ ,213 ,63 ,31 ,63 ,119 ,73 ,41
 ,173 ,59 ,223 ,219 ,74 ,73 ,9
,213 ,223 ,219 ,19 ,0
2060  DATA   0 ,8 ,109 ,73 ,209
 ,255 ,31 ,191 ,77 ,45 ,45 ,213
 ,27 ,63 ,63 ,119 ,73 ,45 ,173
,219 ,219 ,155 ,109 ,73 ,137 ,2
19 ,27 ,63 ,55 ,0 ,0 ,0
2070  DATA   0 ,40 ,45 ,45 ,77
,218 ,63 ,63 ,31 ,110 ,109 ,109
 ,26 ,63 ,255 ,31 ,110 ,41 ,45
,173 ,27 ,63 ,63 ,63 ,46 ,45 ,4
█ ,109 ,218 ,59 ,223 ,55
2080  DATA   0 ,40 ,109 ,109 ,2
09 ,219 ,31 ,63 ,46 ,109 ,73 ,2
09 ,219 ,219 ,51 ,77 ,73 ,137 ,
219 ,219 ,155 ,9 ,77 ,73 ,218 ,
219 ,59 ,55 ,0 ,0 ,0 ,0
2090  DATA   0 ,72 ,105 ,73 ,21
8 ,223 ,251 ,10 ,77 ,109 ,209 ,
223 ,251 ,119 ,77 ,109 ,209 ,25
1 ,27 ,159 ,73 ,77 ,137 ,219 ,2
19 ,155 ,0 ,0 ,0 ,0 ,0 ,0
2100  DATA   0 ,72 ,9 ,77 ,209
 ,27 ,223 ,187 ,9 ,109 ,105 ,26
,255 ,223 ,115 ,41 ,77 ,141 ,21
█ ,223 ,187 ,73 ,105 ,137 ,219
,219 ,155 ,0 ,0 ,0 ,0 ,0
2110  DATA   0 ,72 ,73 ,109 ,21
8 ,255 ,31 ,55 ,45 ,45 ,77 ,213
 ,63 ,63 ,255 ,42 ,45 ,77 ,137
,219 ,219 ,155 ,73 ,73 ,173 ,59
 ,63 ,223 ,19 ,0 ,0 ,0
2120  DATA   0 ,72 ,45 ,45 ,173
 ,251 ,63 ,63 ,87 ,109 ,109 ,21
3 ,31 ,31 ,63 ,55 ,45 ,45 ,77 ,
213 ,63 ,63 ,63 ,87 ,45 ,45 ,45
 ,213 ,255 ,59 ,159 ,0
2130  DATA   █ ,72 ,109 ,45 ,21
3 ,63 ,31 ,223 ,74 ,73 ,45 ,213
 ,255 ,219 ,83 ,73 ,73 ,213 ,21
9 ,219 ,83 ,73 ,105 ,209 ,63 ,2
23 ,155 ,0 ,0 ,0 ,0 ,0
2140  DATA   0 ,8 ,109 ,73 ,209
 ,219 ,219 ,23 ,109 ,73 ,137 ,2
19 ,219 ,63 ,46 ,109 ,73 ,209 ,
219 ,219 ,55 ,109 ,9 ,77 ,218 ,
59 ,223 ,55 ,0 ,0 ,0 ,0
2150  DATA   0 ,40 ,45 ,13 ,77
,218 ,251 ,27 ,55 ,109 ,73 ,141
 ,59 ,63 ,63 ,63 ,110 ,73 ,73 ,
218 ,219 ,27 ,55 ,45 ,77 ,73 ,2
18 ,219 ,63 ,55 ,0 ,0
2160  DATA   0 ,40 ,45 ,45 ,109
 ,26 ,63 ,63 ,63 ,55 ,45 ,13 ,
 45 ,173 ,59 ,63 ,31 ,63 ,46 ,
 109 ,73 ,209 ,219 ,219 ,55 ,
 77 ,73  ,137 ,219 ,219 ,27 ,6
2170  DATA   0 ,72 ,73 ,73 ,218
 ,219 ,219 ,74 ,73,73 ,218 ,22
3 ,219 ,74 ,73 ,109 ,218 ,63 ,2
23 ,83 ,73 ,41 ,141 ,27 ,31 ,22
3 ,19 ,0 ,0 ,0 ,0 ,0
2180  DATA   0 ,72 ,45 ,77 ,
209 ,219 ,59 ,63 ,46 ,77
,73 ,209  ,219 ,219 ,83
,73 ,73 ,209 ,219
,219 ,83 ,73 ,73
,209 ,219 ,21
9 ,19 ,0 ,0
,0 ,0 ,0
```

ANIMAÇÃO GRÁFICA NO TRS-COLOR

Com os comandos GET e PUT podemos armazenar figuras em matrizes e, mais tarde, trazê-las de volta à tela. Aprendendo a utilizá-los, você animará os mais variados desenhos.

Se você já sabe elaborar gráficos com UDG (Caracteres Definidos pelo Usuário), é quase certo que queira movê-los pela tela. Para tanto, será necessário utilizar os comandos **GET** e **PUT**, que já vimos muitas vezes na seção *Programação de Jogos*.

Poderíamos imaginar o **GET** como um comando capaz de "fotografar" uma área retangular da tela de alta resolução do TRS-Color. O que estiver naquela área — um UDG ou qualquer outro gráfico feito com **LINE**, **CIRCLE** ou **DRAW** — será armazenado numa matriz (veja artigo na página 192).

PUT é o oposto de **GET**: ele "coloca" a fotografia (conteúdo da matriz) em qualquer lugar da tela. Usar esse comando é muito mais rápido que redesenhar o gráfico por outros métodos e, com ele, a idéia de movimento pode ser facilmente simulada: basta colocar (**PUT**) uma imagem em diferentes posições na tela ou, então, colocar imagens diferentes repetidamente.

BICICLETA

Digite e rode este programa; ele usa **GET** e **PUT** para movimentar um UDG de uma bicicleta.

```
10 PMODE 4,1:PCLS
20 DIM BY(5)
30 FOR K=1536 TO 1856 STEP 32
40 READ A,B,C
50 POKE K,A:POKE K+1,B:POKE K+2
,C
60 NEXT
70 SCREEN 1,1
80 GET (1,0)-(18,11),BY,G
90 PCLS
100 FOR K=0 TO 238
110 PUT (K,90)-(K+17,100),BY,PS
ET
120 NEXT
130 GOTO 90
140 DATA 1,193,192,0,129,32,0,2
55,64,1,134,0,14,139,128,19,86,
64
150 DATA 36,233,32,47,233,32,32
,104,32,17,100,64,14,3,128
```

Os **DATA** (dados) das linhas 140 e 150 são acessados e depois inseridos na tela, via comando POKE, nas linhas 30 a 60. A bicicleta é desenhada no canto superior esquerdo, ou seja, no começo da tela. Quando você for inserir um grá-

- ANIMAÇÃO DE UM CARACTERE DEFINIDO PELO USUÁRIO (UDG)
- O USO DE MATRIZES COM OS COMANDOS GET E PUT
- "FOTOGRAFE" A TELA
- COMO DIMENSIONAR CORRETAMENTE UMA MATRIZ
- COLOQUE O GRÁFICO ONDE QUISER
- TÉCNICAS DE ANIMAÇÃO

fico na tela e depois usar **GET** e **PUT**, desenhe-o nessa posição, pois assim saberá exatamente onde ele está. Converter posições de memória em posições de tela pode não ser fácil, devido a variações entre os **PMODE**.

A linha 80 faz um **GET** (ou "tira uma fotografia") da bicicleta. A linha 90 limpa a tela antes que as linhas 100 e 120 movimentem a bicicleta. A linha 110 **PUT** ("coloca") a bicicleta em uma posição diferente cada vez que passa pelo laço **FOR...NEXT**.

[illegible] DIMENSIONAR UMA MATRIZ

Sempre que usar **GET** num programa, você precisará dimensionar (**DIM**) uma matriz para armazenar as informações sobre o gráfico. Para calcular o tamanho da matriz faça o seguinte:

1) Depois de desenhar o gráfico, de preferência em papel quadriculado, trace um retângulo, "enquadrando" inteiramente a figura. Conte quantos quadrados existem na primeira linha e quantos existem num dos lados. Multiplique estes dois números — o resultado será o número de quadrados que o desenho ocupa. Por exemplo: o retângulo que envolve a bicicleta tem dezoito quadrados de largura por doze quadrados de altura. Multiplicando-os, obtém-se o número total de quadrados, 216.

2) Em seguida, calcule o número de bytes de memória necessário para armazenar todos [illegible] quadrados. Em **PMODE** [illegible] 4 divide-se o número de quadrados por 8, em **PMODE** 1 e 2 divide-se por 16, e em **PMODE** 0 por 32. Como a bicicleta foi desenhada em **PMODE** 4, divide-se 216 por 8, obtendo-se 27. O resultado deve ser um número inteiro de bytes e, por isso, às vezes é preciso ajustá-lo. E, seja qual for o número obtido, temos sempre que arredondá-lo para cima, nunca para baixo.

3) Para calcular o tamanho que deve ter a matriz, divida o número de bytes por 5 (o divisor é sempre 5, não importa o **PMODE** que está sendo usado).

Voltando ao exemplo da bicicleta: temos 27 bytes para armazenar na matriz; portanto, 27/5 = 5,4. Novamente, [illegible] resultado não for inteiro, deve-se arredondá-lo para cima — 6, [illegible] caso. A linha 20 diz **DIM BY (5)** porque as matrizes começam em 0.

Aqui está um resumo dos divisores necessários para calcular o tamanho das matrizes nos programas gráficos:

Para calcular o número de:

PMODE	Bytes	Matrizes
4	8	5
3	8	5
2	16	5
1	16	5
0	32	5

Pode-se usar **GET** com todo tipo de gráfico na tela de alta resolução do computador. Quer utilizemos um UDG, os comandos **PSET** e **PRESET**, **DRAW**, ou qualquer combinação de comandos gráficos, será sempre possível armazenar figuras em matrizes por meio de **GET** e colocá-las (**PUT**) de volta à tela mais tarde.

Para armazenar um gráfico numa matriz previamente dimensionada, você precisará dizer ao computador onde encontrar o gráfico na tela e em qual matriz pretende armazená-lo. Vejamos esta linha do programa da bicicleta:

```
[illegible] GET(1,0)-(18,11),BY,G
```

Os números nos parênteses são as coordenadas de cantos diagonalmente opostos do retângulo que envolve o gráfico. Você pode especificar os cantos que quiser e em qualquer ordem, desde que sejam diagonalmente opostos.

A área armazenada não compreende todo o UDG que definimos, [illegible] apenas as partes que realmente contêm a imagem — porque a bicicleta não ocupa toda a extensão de 24 quadrados. Como o **GET** não nos limita a mover apenas imagens de oito por oito, podemos movimentar o que bem entendermos.

Na continuação da linha 80, **BY** manda o computador armazenar o gráfico na matriz **BY**, **DIM**ensionada na linha 20.

G significa "detalhe Gráfico completo". Pode-se omiti-lo em certos [illegible] (que explicaremos depois), mas em geral é aconselhável usá-lo.

Se estiver usando UDG, convém sempre inseri-los (**POKE**) no canto superior esquerdo da tela, como foi dito anteriormente, para facilitar o ajuste do **GET**. Lembre-se de que quando os inserimos na tela a posição da imagem não está contida em coordenadas comuns. O **GET** "fotografa" a imagem de uma certa coordenada de tela e, por isso, será preciso converter a posição de memória na qual introduzimos os UDG em coor-

denadas de tela equivalentes. Essa conversão não é fácil, já que varia conforme o **PMODE**. A inserção posições de memória do começo da tela (a partir da posição 1536) facilita cálculo da coordenada de tela do UDG. O canto superior esquerdo deve estar na coordenada (0,0) e, sabendo o tamanho do gráfico, será simples calcular as coordenadas do canto inferior direito do UDG quando "fotografarmos".

O USO DO PUT

Depois de empregarmos o **GET** para armazenar o gráfico na memória, devemos limpar a tela com **PCLS**, antes de colocá-lo (**PUT**) de volta na tela.

Aqui está a linha do **PUT** no programa da bicicleta:

```
110     PUT(K,90)-(K+17,100),BY,PSET
```

Dentro dos parênteses, como no **GET**, encontram-se as coordenadas dos cantos diagonalmente opostos que especificam a área na qual queremos colocar a imagem. **BY** é matriz que contém a bicicleta.

A última parte da linha é **PSET**, que faz computador colocar gráfico na tela exatamente como foi desenhado, sobrepondo-o que estiver naquelas posições.

Existem, porém, outras opções. **PSET** pode ser substituído por **PRESET**, **OR**, **AND**, ou **NOT**, que nos permitem manipular o gráfico.

PRESET diz computador para colocar o gráfico na tela no modo inverso. No modo de duas cores, estas se invertem; de quatro (modo 0), o vermelho fica verde, o azul fica amarelo vice-versa; no modo 1, o laranja fica cinza, o ciano fica roxo e vice-versa.

OR permite que coloque o gráfico armazenado na matriz junto ao que já está na tela. Ambos permanecem inalterados, o que não ocorre quando utilizamos **PSET**, que elimina o que estiver na tela.

AND mostra a junção do que já está na tela com o gráfico que se introduz naquela posição.

NOT não mostra nenhum conteúdo da matriz: simplesmente inverte área da tela em que a matriz será colocada.

A seguir, você verá a utilização desses comandos em um programa.

T

```
10 PMODE 4,1
20 DIM C(5),B1(5),B2(5)
30 PCLS
40 CIRCLE (7,7),7,1:PAINT (7,7),1,1
50 GET (0,0)-(14,14),C,G
60 PUT (0,0)-(14,14),C,NOT
70 GET (0,0)-(14,14),B2,G
80 PCLS1:SCREEN 1,0
90 LINE(8,191)-(104,0),PRESET:LINE(24,191)-(120,0),PRESET:PAINT(10,191),0,0
100 LINE (104,191)-(200,0),PRESET:LINE(120,191)-(216,0),PRESET:PAINT(110,191),0,0
110 FOR K=175 TO 0 STEP -10
120 X=14+(175-K)/2
130 PUT (X,K)-(X+14,K+14),C,OR
140 PUT(X+96,K)-(X+110,K+14),C,OR
150 FOR J=1 TO 100:NEXT
160 PUT(X,K)-(X+14,K+14),B1,PSET
170 PUT(X+96,K)-(X+110,K+14),B2,AND
180 NEXT
190 GOTO 80
```

Se observarmos bem as duas bolas subindo pela tela, veremos que sobram "pontas" ao longo das faixas pretas. Isso acontece porque estamos tentando colocar um gráfico retangular dentro de um inclinado e, cada vez que programa introduz um quadrado preto (lacuna) para "apagar" a posição anterior da bola, as "pontas" aparecem.

Com a bola que caminha pela faixa da direita não ocorre esse problema. Usando uma combinação de **PUT...,AND** e **PUT...,OR**, pontas são eliminadas. A linha 60 inverte a bola na tela e, depois, a linha 70 armazena-a memória. Quando o programa movimenta a bola, linha 140 coloca uma lacuna sobre a faixa com um **PUT...,OR**, de maneira que a bola anterior seja apagada por uma "figura" de formato exatamente igual dela. Em seguida, um **PUT...,AND** coloca na tela a bola invertida. A combinação de **NOT** com **AND** permite que a linha 170 introduza a bola na tela superposição.

Se você quiser colocar um gráfico na tela, num espaço que não seja retangular (um círculo num triângulo, por exemplo), desenhe primeiro o objeto. Coloque-o de volta à tela com **PUT...,NOT** e "fotografe-o" (**GET**) numa matriz. Mais tarde, para mostrar o gráfico na tela, use **PUT...,AND**.

QUANDO NÃO G

Dissemos anteriormente que, em certos casos, pode-se omitir o **G** do final do comando **GET**. Para saber quando omiti-lo, você precisa de algumas informações sobre tela do computador.

Nos **PMODE** 1, 3 e 4 a tela gráfica é dividida em 32 colunas verticais, cada uma com oito pontos de largura, enquanto que nos **PMODE** 0 e 2 a tela é dividida em dezesseis colunas de dezesseis pontos de largura cada. Qualquer que seja o **PMODE** usado, a largura de cada coluna na tela corresponde a um byte memória.

Pode-se omitir **G** do comando **GET** quando o gráfico desenhado ocupa um número determinado de colunas e se quer movimentá-lo. Sem o **G**, bytes inteiros são armazenados matriz declarada. Se o gráfico sobrepuser parte de colunas, você precisará armazenar partes de bytes na matriz. Em outras palavras, será preciso, de alguma maneira, armazenar bits da tela na matriz — e é esta a função do **G**.

Retirando **G**, você terá problemas com gráficos que sobrepõem outras colunas. Além disso, não poderá **PSET**, **PRESET**, **OR**, **AND** ou **NOT** no **PUT** correspondente. A omissão do **G**, portanto, torna menos flexível o uso do **PUT**.

ÇÃO GET E PUT

O **GET** o **PUT** são especialmente úteis nos programas de animação gráfica. Esta é feita, em geral, por meio de dois métodos.

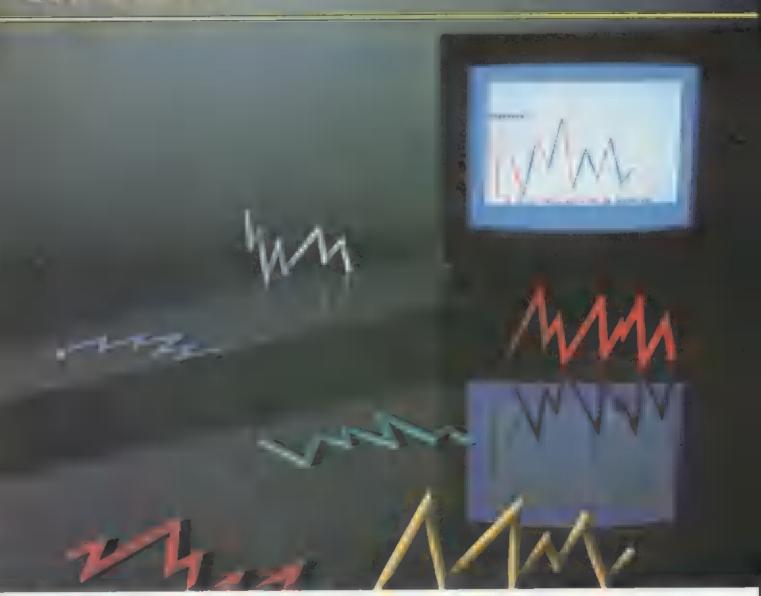
O primeiro deles consiste em usar uma matriz vazia (lacuna) para apagar o gráfico, antes de colocá-lo em outra posição na tela. Para evitar problemas, verifique antes se o tamanho da lacuna é o mesmo do gráfico. Esse procedimento já foi visto em vários artigos de *Programação de Jogos*.

O segundo método, empregado animação da bicicleta, utiliza uma moldura de pontos vazios ao redor do gráfico. Depois que se decide quantos pontos gráfico se moverá a cada passo, verifica-se se existe o mesmo número de fileiras de pontos rodeando o gráfico. Isso significa que o novo gráfico colocado na tela apaga anterior. Se o gráfico movesse para todas as direções, quatro pontos cada passo, por exemplo, precisaríamos de uma moldura com quatro pontos de largura rodeando-o. No caso da bicicleta, o gráfico se move um ponto de cada vez, da esquerda para direita somente. Assim, quando ele foi armazenado, deixamos uma simples fileira de pontos vazios em seu lado esquerdo.

Como exercício, tente animar outros desenhos usando **GET** e **PUT**. A única maneira de entender e se familiarizar com o uso dos comandos **PSET**, **PRESET**, **OR**, **AND** e **NOT** é experimentá-los em diversos gráficos e situações.

COMO TRAÇAR GRÁFICOS

- COMO TRAÇAR GRÁFICOS DE FUNÇÕES MATEMÁTICAS
- CURVAS SERRILHADAS
- OS EIXOS CARTESIANOS
- COMO DETERMINAR ESCALAS

Junte a capacidade de ■ ■ dados com ■ funções gráficas do ■ micro ■ desenvolva programas capazes ■ mostrar em um gráfico qualquer tipo ■ informação numérica.

Armazenamento ■ processamento de dados são atributos típicos de qualquer computador. Contudo, não é fácil para ■ usuário assimilar grandes quantidades de informações. Se ■ apresentarem, por exemplo, uma lista contendo milhares de números, é muito difícil interpretar seu significado.

Para entender melhor, dê ■ olhada na seguinte lista: 132.9, 146.2, 132.89, 123.92, 147.01, 153.47, 132.09, 138.79, 147.57, 153.9, 140.04, 142.76, 152.76, 132.6, 135.09, 146.98. Agora responda rapidamente ■ seguintes perguntas: qual é ■ maior número da lista? Qual ■ sua posição na lista? Quantos números menores que 135.5 há nela?

Como você vê, não é muito fácil dar respostas rápidas ■ essas questões. Imagine ■ a quantidade de números fosse cinco vezes maior!

A maneira tradicional de tratar ■ tipo de problema consiste em colocar ■ informação em um gráfico. Quando isto é feito, torna-se fácil perceber qual é o ponto mais alto. Assim, com o auxílio de uma régua ■ posição horizontal, podemos verificar quais os valores que estão acima ou abaixo de ■ determinado valor.

No artigo *Reúna Seus Dados em Gráficos* (página 181), apresentamos um programa que mostra, sob a forma de um gráfico de barras, dados introduzidos [illegible] computador. O presente artigo trata das técnicas necessárias à confecção de gráficos lineares. Em lições posteriores, abordaremos os histogramas [illegible] outros tipos de gráfico.

Como [illegible] rotinas deste artigo vão além das capacidades gráficas do ZX-81 e do TRS-80, não apresentaremos programas para tais micros.

[illegible] DE [illegible] MATEMÁTICAS

A estrutura de um programa depende do tipo de informação que queremos mostrar na tela. Quando [illegible] [illegible] relação matemática ligando [illegible] dados, podemos usar uma função matemática. Os microcomputadores são dotados, por exemplo, de funções trigonométricas — seno, cosseno e tangente — que tornam fácil o traçado de curvas cíclicas, como já foi explicado em artigos anteriores, ou como demonstra o seguinte programa:

S

```
30 PLOT 0,76
40 FOR t=0 TO PI*10 STEP .3
50 DRAW 2,COS t*15
60 NEXT t
```

MSX

```
10 COLOR 1,15,1
[illegible] SCREEN [illegible]
30 DRAW "BM0,95"
[illegible] FOR T=0 TO 40*ATN(1) STEP .0
[illegible]
50 LINE -(8*T,95-40*SIN(T))
60 [illegible]
150 GOTO 150
```

Apple

```
10  HOME : [illegible] : HCOLOR= 3
30  HPLOT 0,145
40  FOR T = 0 TO 279 STEP [illegible]
[illegible]  HPLOT  TO T,95 + [illegible] *  COS
(T / 10)
60  NEXT T
150  [illegible]
```

T

```
15 PMODE 3,1
20 PCLS
25 [illegible] 1,1
30 DRAW"BM0,95"
40 FOR T=0 TO 40*ATN(1) STEP.06
50 LINE-(8*T,95-40*SIN(T)),PSET
60 NEXT
150 GOTO 150
```

Esse programa traça [illegible] curva senóide, que tem muitas aplicações em ciência, tecnologia, música e traçados gráficos. A linha 40 estabelece que a curva terá cinco ciclos (menos no Apple). Cada ciclo é duas vezes PI. A linha 50 determina como é desenhada a curva (para conhecer maiores detalhes, veja o artigo *Mais Requinte em Seus Desenhos*,

■ página 354). Alterando ■ valores nessas linhas (linha 50 ■ Apple), podemos variar consideravelmente ■ forma da curva.

O micro tem um número limitado de funções embutidas, prontas para serem usadas, mas qualquer função matemática pode ser desenhada, desde que ■ definamos dentro de ■ programa.

Acrescente as próximas linhas ao programa e execute-o novamente.

```
 70 PLOT 0,60
 80 FOR n=1 TO 220 STEP 5
 90 DRAW 5,((110-n)*15)/200
100 NEXT n
110 PLOT 0,100
120 FOR n=1 TO 220 STEP 5
130 DRAW 5,-((110-n)*15)/200
140 NEXT n
```

```
15 D=1:E=64
70 COLOR ■
80 DRAW"BM0,"+STR$(INT(109*D+E)
)
90 FOR T=-127 TO 128
100 LINE -(127+T,T*T/150*D+E)
110 NEXT
120 IF D=-1 THEN 150
130 D=-1:E=E+64:COLOR 6
140 GOTO 80
```

```
5 ■ = 1:E = 64
70  HCOLOR= 5
80  HPLOT 0,64 * ■ + ■
90  FOR T =  - 139 TO 140
100  HPLOT  TO 139 + T,T * T /
160 * D + E
110  NEXT
120  IF D =  - 1 THEN 150
130 D =  - 1:E = E + 64: HCOLOR
= 6
140  GOTO 80
```

```
10 D=1:E=64
70 COLOR 3
■ DRAW"BM0,"+STR$(INT(109*D+E)
)
90 FOR T=-127 TO 128
100 LINE -(127+T,T*T/150*D+E),P
SET
110 NEXT
120 IF D=-1 THEN 150
130 D=-1:E=E+64:COLOR 2
140 GOTO 80
```

Temos agora duas parábolas, além da senóide. Os números necessários para sua elaboração são criados em um laço **FOR...NEXT**. O Spectrum, por exemplo, ■ ■ laço para as parábolas e outro para a senóide.

GRÁFICOS IRREGULARES

A maioria dos gráficos, porém, está vinculada ■ números sem qualquer relação matemática entre si. Esses números podem representar, por exemplo, os valores da precipitação pluviométrica ao longo do ano ou a flutuação dos lucros e perdas de ■ firma: esse tipo de dado varia imprevisivelmente. Se fôssemos traçar o gráfico a mão, começaríamos pelos eixos cartesianos. Assim, façamos com que esta seja ■ primeira parte do programa:

```
140 ■ 0,175
150 PLOT 0,0
160 DRAW 255,0
```

```
5 COLOR 1,15,15
10 SCREEN ■
140 LINE (8,0)-(8,191)
160 LINE (8,191)-(255,191)
200 GOTO 200
```

```
10  ■ : HGR : HCOLOR= 3
140   HPLOT 0,0 TO 0,159
160   HPLOT 0,159 TO 279,159
200   ■
```

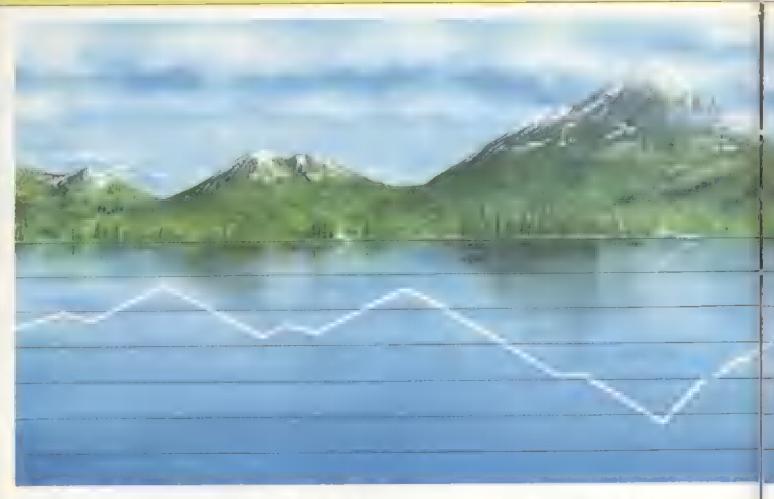
```
5 PMODE 3,1
10 PCLS
15 SCREEN 1,1
140 LINE (0,0)-(0,191),PSET
160 LINE (0,191)-(255,191),PSET
200 GOTO 200
```

Nessas linhas, o programa seleciona um modo gráfico (menos no Spectrum, onde isso não é necessário), colocando o eixo Y no lado esquerdo da tela, e o eixo X, embaixo.

Esta é ■ disposição usual dos eixos, mas podemos modificá-la conforme o intervalo ■ que pertencem os números em questão. Por exemplo, se traçarmos curvas ■ respeito da variação de temperaturas ou sobre ganhos e perdas, teremos (em alguns casos) ■ ocorrência de números negativos; dessa forma, ■ eixo X terá que se deslocar para uma posição mais alta ■ tela.

Outra razão para mudar a disposição dos eixos ■ deixar espaço para números e legendas, sempre úteis quando se trata de gráficos. Faça as próximas modificações para ter um exemplo.

```
130 PLOT 20,10
140 DRAW 0,155
```

150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275

Posso modificar os programas de modo ■ misturar dados positivos ■ negativos ■ ■ só gráfico?

A rotina ■ entrada de dados não precisa ■ modificada para aceitar números negativos; em contrapartida, as rotinas que cuidam dos eixos e dos gráficos exigem essa alteração. A maior dificuldade não ■ descobrir os valores máximo ■ mínimo, mas sim decidir em que ponto da tela colocar o eixo X.

Se você se dispuser ■ mudar a posição desse eixo a cada novo grupo de dados, ■ modificações serão bastante simples. Para isso, observe atentamente os dados, escolha fatores de escala adequados ■ comece a desenhar ■ partir da origem dos eixos.

```
150 PLOT 20,10
160 DRAW 235,0
```

```
5 COLOR 6,15,15
135 COLOR 1
140 LINE (20,0)-(20,160)
160 LINE (20,80)-(255,80)
```

```
10  HOME : HGR
135  HCOLOR= 3
140  HPLOT 20,0 TO 20,140
160  HPLOT 20,70 TO 279,70
200  END
```

```
5 PMODE 3,1
10 PCLS
15 SCREEN 1,1
135 COLOR 2
140 LINE (20,0)-(20,160),PSET
160 LINE (20,80)-(255,80),PSET
200 GOTO 200
```

Ao executar o programa, notaremos que há margens à direita ■ abaixo do gráfico.

Para modificar o tamanho dessas margens, altere os valores nas linhas 130 a 160, com o cuidado de que, ■ Spectrum, ■ linhas 130 e 150 sejam idênticas.

Agora, acrescente ■ próximas linhas para continuar ■ acompanhe o desenvolvimento do processo.

```
  60 LET n=10
  70 PLOT 20,n
  80 FOR x=1 TO 12
  90 READ y
 100 DRAW 18,y-n
 105 LET n=y
 110 NEXT ■
1000 DATA 50,70,60,100,80,120,1
00,130,70,140,90,110
```

```
70 DRAW "BM20,160"
80 FOR X=20 TO 20+11*20 STEP 20
90 READ Y
100 LINE-(X,161-Y)
110 NEXT
160 LINE (20,160)-(255,160)
1000 DATA 140,123,148,45,100,10
,20,8,45,8,45,30
```

```
15  HCOLOR= 5
70  HPLOT 20,160
80  FOR X = 20 TO 20 + 11 * 20
STEP 20
90  READ Y
100  HPLOT  TO X,141 - Y
110  NEXT
160  HPLOT 20,140 TO 279,140
1000  DATA  130,113,138,35,90,5
,20,8,45,8,45,30
```

```
70 DRAW"BM20,160"
80 FOR X=20 TO 20+11*20 STEP 20
90 [illegible] Y
100 LINE -(X,Y),PSET
110 [illegible]
160 [illegible] (20,160)-(255,160),PSE
T
1000 DATA 140,123,148,45,100,10
,20,8,45,8,45,30
```

O programa desenha doze pontos na tela, tomando os valores de X da linha 80, [illegible] os de Y da linha 1000; em seguida, ele une esses valores para formar um gráfico serrilhado.

O primeiro valor da linha **DATA** é desenhado [illegible] extrema esquerda, em cima do eixo dos Y, mas poderíamos tê-lo colocado na primeira posição dos X, modificando, assim, a localização inicial do cursor.

Em todos os programas, menos no do Spectrum, essa localização é especificada na linha 70; no Spectrum, isso é feito invariavelmente na linha 60.

Se os valores vão ser usados várias vezes, convém distribuí-los em uma linha **DATA**. Mas, se quisermos traçar gráficos com valores diferentes, é melhor usarmos o comando **INPUT**.

Acrescente então a próxima linha. Isso funciona em todos os gêneros de microcomputadores, à exceção dos que são compatíveis com o MSX, que não aceita o comando **INPUT** quando está no modo gráfico.

```
90 INPUT "Valor de Y ?",y:
LET y=y+10
```

```
90  VTAB 22: INPUT "VALOR DE Y
?";Y
```

```
90 INPUT "VALOR DE Y ";Y
130 SCREEN 1,1
```

Ao ser executado, o programa pede o primeiro valor de Y. Em seguida, ele realiza [illegible] primeira volta do laço da linha 80. Feito isso, ele torna a pedir um valor de Y, prosseguindo dessa maneira até que os doze valores sejam colocados na tela.

Muitos usuários não gostam da interrupção que [illegible] rotina causa no traçado do gráfico. As seguintes modificações resolvem o problema:

```
 20 DIM y(12)
 40 FOR n=1 TO 12
 50 INPUT "Valor de Y?",y(n):
LET y(n)=y(n)+10
 60 NEXT n: CLS
 70 LET n=10
 [illegible] PLOT 20,n
 90 FOR x=1 TO 12
100 [illegible] 18,y(x)-n
105 LET n=y(x)
110 NEXT [illegible]
```

P&R

O que é escalamento?

Escalamento é uma transformação matemática dos eixos de um gráfico, [illegible] modo [illegible] fazer com que os [illegible] extremos (pontos máximo [illegible] mínimo) caibam nos limites de uma ''janela'' no vídeo. Esta pode ocupar todo [illegible] vídeo, ou apenas parte dele; neste último caso, diversos gráficos podem ser distribuídos simultaneamente por diferentes pontos da tela.

Praticamente todos [illegible] gráficos elaborados por programas exigem algum tipo de escalamento. A única exceção são os gráficos cujas coordenadas horizontais [illegible] verticais [illegible] situam no domínio dos números inteiros positivos, e cujos valores máximos em cada eixo não excedem em número de pixels a resolução da tela na [illegible] direção.

Suponhamos, por exemplo, que é necessário colocar em gráfico duas variáveis, X [illegible] Y, onde X varia de um mínimo de 10 a um máximo de 200, e Y varia de um mínimo de 0 a um máximo de 150. Nesse caso, [illegible] o computador a ser usado for da linha Apple II, que tem uma resolução gráfica máxima de 280 pontos na horizontal por 192 pontos na vertical, não precisaremos escalar os eixos X e Y. Esse exemplo, porém, é uma exceção.

Existem ainda casos em que basta escalar apenas um dos eixos. Entretanto, [illegible] maioria [illegible] vezes, ambos [illegible] eixos precisam ser escalados.

Do ponto de vista matemático, o método de escalamento consiste apenas em uma regra de três. A mesma fórmula pode ser aplicada tanto para o eixo horizontal como para o vertical, bastando trocar as variáveis [illegible] por y. A fórmula leva [illegible] conta [illegible] valores mínimo [illegible] máximo do eixo, o número de pontos gráficos que cabem nele, [illegible] [illegible] [illegible] valores numéricos identificadores das marcas dos eixos (rótulos) serão arredondados ou não.

A fórmula de escalamento é:

$$x' = \frac{x - xmin}{[illegible] - xmin} \cdot pix$$

onde:

x' = valor escalado
x = valor original
xmin = valor mínimo da escala
xmax = valor máximo da escala
pix = número máximo de pixels do eixo.

```
5 COLOR 1,15,15
10 SCREEN 0
20 DIM Y(11)
40 FOR N=0 TO 11
50 INPUT "Qual o valor de Y ";Y
(N)
60 NEXT
[illegible] COLOR 6:SCREEN 2
70 DRAW "BM20,"+STR$(INT(161-Y(
0)))
80 N=[illegible]
[illegible] [illegible] X=20 TO 11*20+20 STEP 20
100 LINE -(X,161-Y(N))
110 N=N+1
120 [illegible]
```

```
10  HOME
15  DIM Y(11)
40  FOR N = 0 TO 11
50  INPUT "QUAL O VALOR DE Y ?
";Y(N)
60  NEXT
65  HOME : HGR : HCOLOR= 5
70  HPLOT 20,141 - Y(0)
[illegible] [illegible] = 0
[illegible]  FOR X = 20 TO 11 * 20 + 20
STEP 20
100  HPLOT  TO X,141 - Y(N)
110 N = N + 1
120  NEXT
```

```
20 DIM Y(11)
40 FOR N=0 TO 11
50 INPUT "VALOR DE Y ";Y(N)
60 NEXT
70 DRAW "BM20,"+STR$(INT(161-Y(
0)))
80 N=0
90 FOR X=20 TO 11*20+20 STEP 20
100 LINE-(X,Y(N)),PSET
110 N=N+1
120 NEXT
```

O programa realiza primeiro [illegible] rotina de entrada de dados para depois fazer o gráfico. Os dados são colocados na matriz Y ( ), como variáveis de Y (1) a Y (12) no Spectrum, e Y (0) a Y (11) nos demais computadores. Caso se queira utilizar mais de doze números, basta redimensionar [illegible] matriz. Os valores de X são obtidos na linha 90, e o desenho é feito pela linha 100. A linha 110 conta os valores de Y.

COMO USAR ESCALAS

Os valores para os gráficos foram escolhidos até agora de modo a caber dentro dos eixos [illegible] da tela. Geralmente, porém, os números que queremos colocar em um gráfico são maiores ou menores que o ideal. Precisamos, assim, de uma escala que os coloque dentro do intervalo válido. A maneira mais simples de fazer isso [illegible] olhar todos os valores e decidir então qual é o fator que deveremos usar para multiplicar todos os números, de forma [illegible] mostrá-los no gráfico.

A seguir, apresentamos o programa completo, isto é, com todas [illegible] modificações necessárias.

```
  20 DIM y(12)
  30 LET xs=2: LET ys=1/50
  40 FOR n=1 TO 12
  50 [illegible] y(n)
  60 NEXT n: CLS
  70 LET n=8
  80 PLOT 40,n
  90 FOR x=1 TO 12
 100 DRAW 8*xs,(y(x)-n)*ys
 105 LET n=y(x)
 110 NEXT [illegible]
 130 PLOT 40,8
 140 DRAW 0,167
 150 PLOT 40,8
 160 DRAW 210,0
1000 DATA 4000,2000,6000,3000,8
000,4000,5000,2000,6000,1500,30
00,500
```

```
5 COLOR 6,15,15
10 SCREEN 2
20 DIM Y(12)
30 XS=20:YS=1/20
40 FOR N=1 TO 12
50 READ Y(N)
60 NEXT
[illegible] [illegible] "BM20,"+STR$(INT(161-YS
*Y(1)))
90 FOR X=1 TO 12
100 LINE-(X*XS,161-YS*Y(X))
110 NEXT
135 COLOR 1
140 [illegible] (20,0)-(20,160)
160 LINE (20,160)-(255,160)
200 GOTO 200
1000 DATA 1600,1000,2500,3000,3
200,2000,2500,1000,3000,750,150
0,250
```

```
10  [illegible] : [illegible] : HCOLOR= 5
20  DIM Y(12)
30 XS = 20:YS = 1 / 20
40  FOR N = 1 TO 12
50  READ Y(N)
60  NEXT
80  HPLOT 20,141 - Y(1) * YS
90  FOR X = 1 TO 12
100  HPLOT  TO [illegible] * XS,141 - Y(X
) * YS
110  NEXT
135  HCOLOR= 3
140  HPLOT 20,0 TO 20,140
```

```
160  HPLOT 20,140 TO 279,140
200  END
1000  DATA  1600,1000,2500,300,
450,2000,2500,1000,2700,750,150
0,250
```

```
5 PMODE 3,1
10 PCLS
20 DIM Y(12)
30 XS=20:YS=1/20
40 FOR N=1 TO 12
50 READ Y(N)
60 NEXT
80 DRAW"BM20,"+STR$(INT(161-YS*
Y(1)))
90 FOR X=1 TO 12
100 LINE -(X*XS,Y(X)*YS),PSET
110 NEXT
130 SCREEN 1,1
135 COLOR 2
140 LINE (20,0)-(20,160),PSET
160 LINE (20,160)-(255,160),PSE
T
200 GOTO 200
1000 DATA 1600,1000,2500,3000,3
200,2000,2500,1000,3000,750,150
0,250
```

O programa faz o gráfico a partir dos valores da linha **DATA** 1000. Assim, poupa-se o tempo de digitação dos números enquanto o programa está sendo testado. Como vimos anteriormente, é possível mudar facilmente a via de entrada dos dados.

Os fatores de escala são estabelecidos pela linha 30. A linha 80 move o cursor para a posição inicial e os pontos são traçados pelo laço das linhas 90 a 110. As linhas 130 a 160 desenham os eixos cartesianos.

Se o gráfico não utilizar todo o espaço disponível na tela, podemos mudar o valor dos fatores de escala da linha 30. Para usar a escala nos eixos, digite as próximas linhas.

```
140 DRAW 0,2000*ys
160 DRAW 12*xs,0
```

```
140 LINE (20,160)-(20,161-3200*
YS)
```

```
140  HPLOT 20,140 TO 20,141 - 2
700 * YS
```

```
140 LINE (20,160)-(20,161-3200*
YS),PSET
```

Ao rodarmos o programa, o eixo Y irá somente até a altura do maior valor. O efeito será mais nítido se substituirmos os valores da linha 30. Agora podemos utilizar qualquer valor, desde que modifiquemos os fatores de escala da linha 30.

Para numerar os eixos, digite as próximas linhas:

```
210 FOR x=0 TO 12 STEP 2
220 PRINT AT 21,x*2+4;x
230 NEXT x
300 FOR y=0 TO 8000 STEP 2000
310 PRINT AT (8000-y)/400,0;y
320 NEXT y
```

As linhas 210 a 230 tomam os valores de 0 a 12, que são colocados logo abaixo do eixo X. Os números ao longo do eixo Y são impressos pelas linhas 300 a 320.

```
200 OPEN "GRP:" FOR OUTPUT AS #
1
210 FOR Y=500 TO 3000 STEP 500
220 PRESET (0,151-Y*YS):PRINT#1
,Y/500
230 LINE (18,161-Y*YS)-(20,161-
Y*YS)
240 NEXT
250 PRESET (25,0):PRINT #1,"(x
500)"
260 FOR X=20 TO 255 STEP 20
270 PRESET(X-8,170):PRINT#1,X/2
0
280 LINE (X,162)-(X,160)
290 NEXT
300 GOTO 300
```

A linha 200 abre um arquivo para que possamos escrever na tela de alta resolução. O laço das linhas 210 a 240 escreve os valores do eixo X e o laço das linhas 260 a 290, no eixo Y. Note como **PRESET** é usado para escolher a posição de impressão. A linha 250 informa a escala das ordenadas.

```
200  FOR Y = 500 TO 2500 STEP 5
00
210  HPLOT 18,Y * YS TO 20,Y *
YS
220  NEXT
230  FOR X = 20 TO 20 * 12 STEP
 20
240  HPLOT X,142 TO X,140
250  NEXT
260  END
```

Este programa apenas coloca tracinhos nos eixos. O **Apple** não tem uma maneira de escrever com facilidade na tela de alta resolução.

No TK-2000 podemos escrever os números na tela gráfica; neste caso, porém, a tarefa é deixada ao leitor.

O laço das linhas 200 a 220 cuida do eixo Y, enquanto que o das linhas 230 a 250 cuida do eixo X.

Escrever na tela gráfica do TRS-Color é mais complicado, pois não podemos usar o **PRINT**. Os caracteres devem ser desenhados com **DRAW**. Muito longa para ser listada aqui, a rotina que faz isso encontra-se na página 236 (*Recursos Gráficos Sofisticados*), e pode ser facilmente incorporada ao programa apenas com alguns ajustes nos números das linhas.

COMO COMPARAR DADOS

A capacidade de desenhar um gráfico em escala é particularmente útil quando se deseja mostrar mais de um conjunto de dados ao mesmo tempo. Por exemplo, as duas metades do ano. Se quisermos compará-las, devemos começar desenhando um gráfico para o primeiro semestre no alto da tela e outro para o segundo, embaixo. Podemos colocar os doze valores dentro de uma só matriz. Depois, lemos os seis primeiros e acionamos a sub-rotina que faz o gráfico. Lemos a seguir os outros seis, e repetimos o processo. As mudanças necessárias dizem respeito à posição inicial de cada gráfico e à posição dos eixos, de forma a desenhar dois, e não apenas um eixo. Uma alternativa para essa solução é traçar um único eixo X no meio da tela, de onde partem, em direções diferentes, os dois gráficos.

A ARTE FINAL

Os últimos retoques no desenho ficarão por sua conta. Eles não são essenciais, mas o gráfico ficará mais atraente se for colorido e dispuser de um rodapé. Os comandos necessários para realizar esse trabalho podem ser revisados nos artigos *Como Desenhar em BASIC* (página 113) e *Ordem e Limpeza no Vídeo* (página 146).

CUIDADOS COM FITAS E DISCOS

- PROTEJA A INFORMAÇÃO
- COMO FAZER CÓPIAS CAUTELARES
- INDEXAÇÃO DO MATERIAL
- FAÇA ENVIOS PELO CORREIO

Você não precisa ser um gênio de organização para lidar com computadores. Mas vale a pena fazer um mínimo de esforço para manter em ordem suas fitas e disquetes.

Os sistemas de armazenamento auxiliar baseados em fitas ou discos oferecem ao usuário de computadores domésticos [illegible] possibilidade de contar com vastas quantidades de informação e também de milhares de programas de uma forma muito compacta.

Entretanto, a eficiência dos sistemas de armazenamento magnético tem alguns pontos fracos em potencial. Como as informações se concentram em pequenos disquetes ou em fitas cassetes, qualquer dano sofrido por estes pode ter efeitos desastrosos. Ao mesmo tempo, os meios magnéticos são bastante vulneráveis a agressões ambientais.

PROTEÇÃO CONTRA DANOS

Diante disso, é de fundamental importância que [illegible] informação permaneça inalterada no disco ou na fita em que foi gravada. Assim, você poderá tê-la de volta sempre que precisar.

Existem dois tipos de danos que podem afetar os meios magnéticos: o tipo mecânico e o magnético.

Os danos mecânicos podem ser causados por fatores externos, como calor, deformação (provocada pela pressão de um corpo mais pesado, por exemplo, poeira, umidade etc.). Esses fatores tendem a deteriorar a base magnética ou a estrutura dos disquetes e das fitas. Para evitar sua ação, é necessário proteger os meios magnéticos sempre que eles não estiverem sendo utilizados. Assim, mantenha os disquetes dentro de sua capa de cartolina [illegible] guarde-os [illegible] uma caixa ou gaveta apropriada. Conserve as fitas cassetes em suas caixas de plástico, e estas, por sua vez, em uma estante própria (o que facilitará também o trabalho de localizá-las). Armazene discos e fitas longe da poeira, do calor e da umidade.

Nunca toque [illegible] superfície exposta de um disco ou fita. Rebobine as fitas sempre que puder, colocando sua parte transparente (*leader*) na abertura, onde a fita está mais sujeita à ação dos agentes externos. Além disso, [illegible] fita ficará pronta para ser novamente utilizada. Não deixe fitas por muito tempo dentro do gravador com a tecla **PLAY** acionada; [illegible] o fizer, elas sofrerão deformações provocadas pelo cabeçote do gravador. Da mesma forma, nunca deixe disquetes dentro da unidade acionadora (*driver*).

Outro perigo considerável para seus discos e fitas são os campos magnéticos encontrados em aparelhos eletrodomésticos como alto-falantes e alguns tipos de motor. Equipados com ímãs, esses aparelhos geram campos magnéticos fortes mesmo quando não estão funcionando. Portanto, mantenha discos e fitas longe de televisores, motores de automóveis, geladeiras, aparelhos de ar condicionado, ventiladores, e até mesmo de telefones.

[illegible] CONTRA PERDAS

Por outro lado, os acidentes são inevitáveis; por isso, vale [illegible] pena tomar precauções extras e fazer duplicatas dos dados e programas mais importantes. Essas cópias cautelares, ou de segurança, são conhecidas em computês pelo nome de *back-up*.

Em geral, é uma boa idéia gravar tudo duas vezes (o que pode ser feito na mesma fita ou disco). Três vezes é melhor ainda. Desse modo, pelo menos uma versão permanecerá intacta. Se o arquivo for importante, faça cópias em novos discos ou fitas, [illegible] guarde-as em outro lugar; mas procure não utilizá-las: elas poderão ser necessárias para recuperar [illegible] arquivo original em caso de perda.

Proteja suas fitas contra [illegible] desgravação, removendo a orelha de plástico da borda da fita. Para proteger os disquetes, coloque [illegible] fita adesiva opaca [illegible] reflexiva sobre o furo retangular que existe em uma ou ambas as bordas (isso, nos disquetes de 5 1/4". No [illegible] dos disquetes de 8", a proteção se faz removendo o adesivo).

Com o passar do tempo, discos e fitas se acumularão às centenas em seu arquivo, tornando cada vez mais difícil a localização dos dados e programas. Assim, é aconselhável rotular discos e fitas desde o começo (você pode, por exemplo, numerá-los para referências rápidas; alguns micros permitem identificar disquetes com nomes). Paralelamente, procure armazenar apenas arquivos relacionados em uma fita ou disco. As fitas devem, de preferência, ser curtas e conter de um a três arquivos ou programas, apenas.

Dê [illegible] cada arquivo um nome bem claro, dentro das limitações impostas pelo sistema operacional usado pelo seu micro. Se você tiver várias versões de um programa, utilize um sistema de numeração qualquer para identificá-las. Coloque o nome [illegible] [illegible] número de cada versão dentro dos programas utilizando declarações **REM**.

Para identificar discos e fitas, use etiquetas gomadas adequadas. No caso de fitas, não se esqueça de rotular também as caixas. Não escreva com lápis ou caneta esferográfica sobre [illegible] etiqueta de um disquete. Mantenha um caderno ou fichário com todos os nomes de arquivos, programas, comentários e suas respectivas localizações.

PELO CORREIO

Fitas e discos constituem um meio muito conveniente de enviar programas e outros tipos de informação pelo correio. As fitas cassetes são razoavelmente fortes e "viajam" bem. Envie-as sem a caixa para reduzir o peso e o custo (não esqueça de travar o movimento das bobinas). Envelopes almofadados ou a caixinha lacrada de fonogramas são ideais para esses envios.

No caso de discos, convém embalá-los entre dois pedaços de papelão rígido para evitar deformações. Se você preferir, pode também colocá-los em caixas metálicas rasas ou de plástico resistente. Em todos os casos, escreva do lado de fora do pacote: "MEIO MAGNÉTICO. FRÁGIL". Para maior segurança, envie registrado.

GERAÇÃO DE BLOCOS GRÁFICOS (1)

- CRIE NOVOS CARACTERES
- COMO COLOCAR PADRÕES EM UM BANCO DE MEMÓRIA
- TECLAS DE CONTROLE
- CORES EM ALTA RESOLUÇÃO

Todos os que já [illegible] [illegible] como é árduo o trabalho de criar [illegible] [illegible] usando papel e lápis. Mas nem tudo [illegible] perdido: [illegible] aqui um programa que [illegible] esse trabalho.

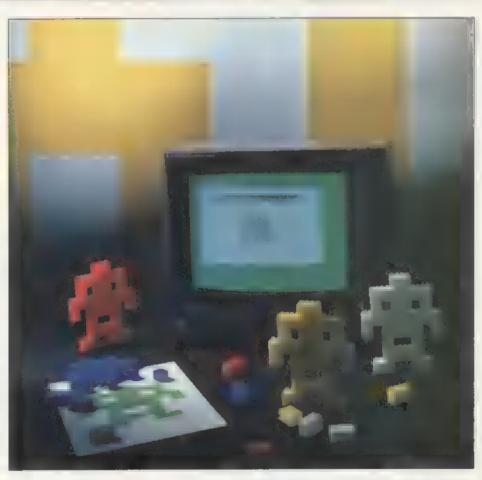
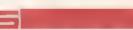
Conhecidos pela sigla UDG, os blocos gráficos são um dos recursos mais úteis ao programador gráfico. O trabalho de planejá-los [illegible] desenhá-los inicialmente [illegible] papel, entretanto, exige muito tempo [illegible] dedicação. Além disso, [illegible] cálculo dos números correspondentes a se[illegible] utilizados em linhas **DATA** é bastante complicado. Por outro lado, se algum erro for cometido, no decorrer do processo, [illegible] correção também exigirá um esforço especial.

O programa apresentado neste artigo facilita [illegible] coisas. Ele substitui o papel milimetrado pela tela do micro, onde poderemos desenhar [illegible] padrão do novo bloco gráfico. O cálculo dos valores a serem colocados em linhas **DATA** também [illegible] feito automaticamente.

Outra grande vantagem do programa [illegible] que ele nos deixa ver, em tamanho natural, o bloco que estamos criando [illegible] nos poupa o trabalho de transferir [illegible] dados para outro programa. Várias teclas de controle nos permitirão não só modificar o padrão do desenho, como também produzir o padrão inverso, ou virá-lo da esquerda para [illegible] direita. Tudo isso com um simples toque.

Quando estivermos satisfeitos com [illegible] padrão criado, poderemos preservá-lo, gravando todo um conjunto de blocos. Uma vez gravado, esse conjunto poderá ser carregado para utilização em outros programas, [illegible] reeditado pelo presente gerador de blocos gráficos.

O programa virá em duas partes. A primeira, neste artigo, cuida do aspecto básico do gerador. Faz um quadriculado [illegible] tela, permitindo a movimentação de um cursor para a criação do padrão desejado. Na parte seguinte, serão apresentadas rotinas que possibilitam modificações mais sutis [illegible] caractere.

Grave a primeira parte em fita ou disco para completá-la posteriormente. Devido à falta de alta resolução gráfica [illegible] ZX-81 e no TRS-80, não mostraremos versões para esses micros.

Colocado em funcionamento, o programa mostra na tela um quadriculado com [illegible] [illegible] [illegible] quadradinhos. Este é o "papel milimetrado" no qual iremos criar nosso novo bloco gráfico. Agora podemos escolher quais quadradinhos "acender" e quais "apagar" para criar um padrão. Para tanto dispomos de um quadradinho branco, que funciona como cursor. Desprovido de cor especial, este último é, na verdade, a versão mais brilhante — com atributo **BRIGHT** — do quadradinho que está abaixo dele. Assim, podemos ver, ao mesmo tempo, onde o cursor está [illegible] se esse quadradinho foi aceso ou apagado. O programa foi escrito de modo que as setas movimentem o cursor e o número 0, [illegible] acendam ou apaguem os quadradinhos.

Quem preferir outras teclas para movimentar [illegible] cursor deve mudar os caracteres dentro do **IF I$ =** ... nas linhas 6520 a 6555.

O mesmo método serve para adaptar o programa ao uso de joysticks. Quem dispuser de um desses periféricos deve consultar [illegible] artigo da página 348.

Aqueles que não quiserem recorrer às setas para mover o cursor terão de escolher as novas teclas com bastante cuidado, pois nenhuma das teclas de controle pode ser usada. Essa parte inicial do programa utiliza apenas três teclas de controle (explicaremos melhor adiante), mas a parte seguinte usa mais. De qualquer modo, [illegible] teclas C, I, M, P, R, S

e T estão proibidas.

Usando o teclado ou o joystick, poderemos movimentar o cursor dentro do quadriculado. Quando pressionarmos a tecla 0, no teclado (ou o botão do joystick, se forem feitas as modificações adequadas), veremos a cor do cursor mudar, indicando que o quadradinho onde ele está foi "aceso". Se alterarmos a posição do cursor, provocaremos uma mudança na cor do quadradinho. Dessa maneira, podemos definir [illegible] padrão de novos blocos gráficos.

O programa oferece várias opções, que podem ser feitas a qualquer momento da edição. Uma delas é a de guardar o bloco no banco de memória. Para tanto, basta pressionar S — uma das teclas de controle que mencionamos antes.

O programa procurará saber então em que lugar do banco de memória deverá guardar o bloco. O banco é representado pelas letras de A a U, mostradas na tela acima do quadriculado. Depois de termos teclado uma letra entre A e U, haverá uma pausa, enquanto os números correspondentes ao padrão do bloco são colocados no banco de memória, usando **POKE.** Logo em seguida, o novo bloco aparecerá na tela, ocupando seu lugar no banco.

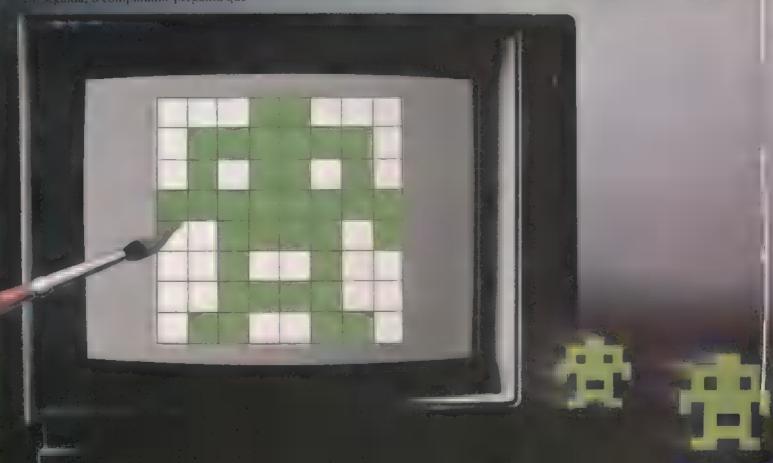
Outra opção consiste em recuperar um bloco guardado e promover sua edição. Para essa opção usamos a tecla P; em seguida, o computador pergunta que bloco desejamos. Em resposta, devemos pressionar [illegible] tecla correspondente; [illegible] bloco desejado [illegible] então colocado no quadriculado.

GRAVE [illegible] NA FITA

Precisamos agora gravar [illegible] conjunto de blocos criados, de modo a utilizá-los em outros programas. Ao [illegible] [illegible]po, é aconselhável carregar um conjunto de blocos para edição.

Para começar, pressionamos a tecla T. O programa perguntará então se queremos gravar ou carregar [illegible] conjunto. No primeiro caso, todo o conteúdo do banco de memória será cuidadosamente gravado; portanto, não devemos esquecer de guardar ali o último caractere que estava sendo editado.

A próxima parte do artigo explicará como usar [illegible] recursos do gerador para criar e utilizar muitos [illegible] muitos blocos gráficos, incluindo como [illegible] [illegible] conjunto criado em outros programas. Novos recursos interessantes serão também adicionados [illegible] programa.

```
 5 CLEAR USR "A"-16
 6 POKE 23675,PEEK 23675-16
 [illegible] BORDER 4: [illegible] 7: INK 0:
 CLS
10 FOR N=USR "A" TO USR "B"+7
 : READ A: POKE N,A: NEXT [illegible]
15 POKE 23675,PEEK 23675+16
 20 [illegible] 23658,8
 30 LET X=11: LET Y=8: LET NX=
 X: LET NY=Y
 100 LET A$="": [illegible] N=144 TO
 164: LET A$=A$+CHR$ N: [illegible] N
 110 LET B$="": FOR N=65 TO 85:
 LET B$=B$+CHR$ N: NEXT N
1000 FOR N=87 TO 151 STEP 8: PL
OT N,48: DRAW 0,64: NEXT [illegible]
1010 FOR N=48 TO 112 STEP 8: [illegible]L
OT [illegible]7,N: [illegible] 64,0: NEXT N
2000 PRINT AT 3,5;A$
2010 PRINT  INVERSE 1;AT 2,5;B$
2400 GOSUB 6500
2500 IF INKEY$="P" [illegible]  GOTO [illegible]
000
2510 IF INKEY$="S" [illegible]  GOTO [illegible]
100
2520 IF INKEY$="T" THEN  GOTO 5
200
3900 GOTO 2000
5000 INPUT "QUE CARACTER (A-U)?
", LINE C$
5010 IF C$<CHR$ 65 OR C$>CHR$ [illegible]
5 THEN  GOTO 5000
5020 LET D=CODE C$+79: LET C$=C
[illegible] [illegible]: GOSUB 6000: GOTO 2000
5100 INPUT "GUARDAR EM QUE LETR
[illegible] (A-U)?", LINE C$
5110 IF C$<CHR$ 65 [illegible] C$>CHR$ [illegible]
5 THEN  G[illegible] 5100
5120 PRINT AT 18,10;"GUARDANDO"
5130 [illegible] N=USR C$ TO [illegible] C$+7
5140 LET R=0: LET BIT=128: FOR
M=0 TO 7
5150 IF PEEK (18432+(N-USR C$)*
32+M+11)<>1 THEN  LET R=R+BIT
5160 LET BIT=BIT/2: NEXT M: [illegible]
```

```
E N.R
5170 NEXT N: PRINT AT 18.0;" ";
TAB 31;" ": GOTO 1000
5200 INPUT "C(ARREGAR) OU G(RAV
AR)?", LINE C$: IF C$<>"C" AND
C$<>"G" THEN  GOTO 1000
5220 IF C$="C" THEN  GOTO 5250
5230 INPUT "INTRODUZA O NOME DO
 ARQUIVO", LINE N$: IF N$="" OR
 LEN N$>10 THEN  GOTO 5230
5240 SAVE N$CODE USR "A",168: G
OTO 100
5250 INPUT "INTRODUZA O NOME DO
 ARQUIVO", LINE N$: IF LEN N$>1
0 THEN  GOTO 5250
5260 PRINT AT 19,0;: LOAD N$COD
E USR "A": PRINT AT 20,0;" ";TA
B 31;" ": GOTO 100
6000 LET B=USR "A"+8*(CODE C$-1
44)
6010 POKE 23675,PEEK 23675-16
6020 FOR N=0 TO 7
6030 LET V=PEEK (B+N)
6040 LET BIT=128: FOR M=0 TO 7
6050 IF V>=BIT THEN  PRINT AT 8
+N,11+M;CHR$ 144: LET V=V-BIT:
GOTO 6060
6055 PRINT AT 8+N,11+M;CHR$ 145
6060 LET BIT=BIT/2: NEXT M
6070 NEXT N: POKE 23675,PEEK 23
675+16: RETURN
6500 POKE 22528+32*Y+X,120: PAU
SE 0
6510 LET I$=INKEY$
6520 IF I$="5" AND X>11 THEN  L
ET NX=X-1
6530 IF I$="8" AND X<18 THEN  L
ET NX=X+1
6540 IF I$="6" AND Y<15 THEN  L
ET NY=Y+1
6550 IF I$="7" AND Y>8 THEN  LE
T NY=Y-1
6552 IF I$="0" THEN  GOSUB 7000
6555 IF I$="" THEN  GOTO 6580
6560 POKE 22528+Y*32+X,56
6570 LET X=NX: LET Y=NY
6580 POKE 22528+Y*32+X,120
6590 RETURN
7000 POKE 23675,PEEK 23675-16
7010 IF PEEK (18432+(Y-8)*32+X)
=1 THEN  PRINT  BRIGHT 1;AT Y,X
;CHR$ 144: GOTO 7030
7020 PRINT  BRIGHT 1;AT Y,X;CHR
$ 145
7030 POKE 23675,PEEK 23675+16:
RETURN
9000 DATA 85,171,85,171,85,171,
85,255,1,1,1,1,1,1,1,255
```

Quando o programa é executado, um quadriculado de 8 x 8 pontos surge na tela. Este será o "papel quadriculado" onde planejaremos a forma de nosso bloco gráfico.

Temos também um cursor em forma de cruz. Conforme as cores escolhidas, o cursor será pouco ou muito visível, podendo até mesmo desaparecer quando sua cor for "transparente" (código de cor zero). O programa foi escrito de maneira que as setas movimentem o cursor e a barra de espaço acenda ou apague o ponto.

Se você quiser utilizar um joystick para mover o cursor (os pontos são acesos pelo botão de disparo), deve mudar o valor das instruções **STICK** e **STRIG**, nas linhas 70 e 20, respectivamente.

Assim, o cursor pode ser movimentado dentro do quadriculado. Quando pressionarmos a tecla de espaço, o ponto que está sob o cursor será aceso, assumindo a cor de frente; se pressionarmos a tecla novamente, ele será apagado, adotando a cor de fundo.

No início do programa, a cor de fundo é branca e a cor de frente, preta. Se quisermos modificar esta última, devemos pressionar a tecla C. O computador nos perguntará a seguir o código da nova cor de frente, ou seja, um número de 0 a 15 em decimal. A escolha dessa cor pode ser feita com apenas um toque: para isso, devemos digitar seu número em hexadecimal (de 0 a F). Já a cor de fundo pode ser modificada pressionando-se a tecla F.

É importante ter em mente as regras de apresentação das cores na tela de alta resolução do MSX. Cada conjunto de oito pontos adjacentes na horizontal, correspondentes a uma linha do nosso bloco gráfico, tem apenas uma cor de frente e uma cor de fundo. Se uma segunda cor de frente for atribuída a um ponto, todos os pontos acesos daquela linha do bloco passarão a ter esta cor. O mesmo acontecerá com os pontos apagados, se tentarmos introduzir uma segunda cor de fundo. Em seu programa, esses cuidados ficam por sua conta; o computador pode mostrar até oito cores numa mesma linha de bloco *no quadriculado*. No momento em que esse bloco for colocado na tela, em tamanho natural, apenas as duas últimas cores — uma de frente e outra de fundo — aparecerão.

Essa limitação no número de cores pode causar inconvenientes. Se uma linha do bloco tiver a cor de frente igual à de fundo, o traçado do quadriculado naquela linha desaparecerá. Neste caso, de pouco valeria modificar o programa, fazendo-o redesenhar o quadriculado; isso, na verdade, criaria novos problemas. Na realidade, o traçado está ali: só não podemos vê-lo, porque ele tem a mesma cor do fundo.

O programa tem ainda outras funções, disponíveis a qualquer momento durante a edição de um bloco. Uma delas "guarda" o bloco em um banco de memória, que fica protegido na parte alta da memória. Para fazer isso, basta apertar a tecla G.

O programa solicita então um número correspondente à posição que o bloco ocupará no banco de memória. Esse número pode valer de 0 a 255. Uma vez informado da posição desejada, o computador mostra o bloco, em tamanho natural, no alto da tela. Ali fica representado todo o conteúdo do banco de memória. Um mesmo bloco pode ser guardado em várias posições diferentes.

Outra opção oferecida é a de "recuperar" um bloco que esteja no banco, trazendo-o de volta ao quadriculado, onde ele pode sofrer modificações. Isto é feito pela tecla R. Note que apenas o desenho do bloco é trazido para o quadriculado. O bloco permanece guardado no banco, e só será apagado quando outro for guardado na mesma posição.

COMO GRAVAR OS BLOCOS EM FITA

Após tanto trabalho, você certamente vai querer gravar os blocos criados para um eventual uso futuro. O programa dispõe, para tanto, de uma função de gravação em fita, acompanhada de outra, que permite a leitura de um banco previamente gravado.

Para gravar ou ler uma fita, é preciso pressionar a tecla T. O computador perguntará então se você deseja um **SAVE** ou um **LOAD**. Se você escolher gravar, o banco de memória será transferido em bloco para a fita. Portanto, não se esqueça de guardar o último bloco que estava sendo editado.

O banco de memória utilizado nada mais é do que uma região da memória do micro, protegida para esse fim. Ela contém, no entanto, alguns valores inúteis para nós (*lixo*, no jargão do programador). Esse "lixo" não é visível durante a execução do programa, mas pode ser transferido para a fita, quando não usamos todas as 256 posições. Ele torna-se visível quando carregamos um banco da fita.

Em geral, as posições múltiplas de 16 apresentam pontos aleatórios. Esse problema, porém, só afeta posições não ocupadas, não interferindo nos blocos. Da mesma forma, se interrompermos o programa por qualquer motivo, e voltarmos a executá-lo, não veremos o banco na tela; mas todos os blocos criados estarão lá, podendo ser recuperados, especialmente quando soubermos suas posições.

Quando quisermos ler um conjunto de blocos da fita, o programa oferecerá a opção de ver seu conteúdo por meio da tecla V. Se soubermos exatamente onde estão os blocos, podemos poupar tempo e trabalho. Nesse caso, a tecla <ENTER> nos trará de volta à edição (não veremos nada na tela, mas todo o conteúdo do banco estará lá, podendo ser recuperado).

A próxima parte deste artigo explicará como utilizar estes e outros recursos do programa para criar blocos gráficos, bem como as formas de utilizá-los em outros programas.

```
10 CLEAR 200,&HD000:SCREEN1:FOR
 I=28*8 TO 28*8+7:A=VPEEK(BASE(
7)+I):A$=A$+CHR$(A):NEXT:COLOR
1,15,15:C=1:F=15:C1=C:C2=F:SCRE
EN 2,0
20 DIM B(8,8):SPRITE$(0)=A$:ON
STRIG GOSUB 500,500:STRIG(0) ON
30 GOSUB 1430
50 X=96:Y=64:GOTO 200
60 K$=INKEY$:IF K$="" THEN 60
70 A=STICK(0)
```

```
T(8,170):PRINT#1,"o número do b
loco :":CLOSE 1:GOSUB 9200
1010 X$=INKEY$:IF X$="" THEN 10
10
1015 IF X$=CHR$(13) THEN GOSUB
9100:GOTO 1050
1020 IF X$=CHR$(29) AND N>0 THE
N N=N-1:GOSUB 9200:GOTO 1010
1025 IF X$=CHR$(28) AND N<255 T
HEN N=N+1:GOSUB 9200:GOTO 1010
1030 IF X$=CHR$(31) AND N>9 THE
N N=N-10:GOSUB 9200:GOTO 1010
1040 IF X$=CHR$(30) AND N<245 T
HEN N=N+10:GOSUB 9200:GOTO 1010
1045 GOTO 1010
1050 FOR I=0 TO 7:P(I)=PEEK(&HD
100+N*8+I)
1052 FOR I=0 TO 7:P(I)=PEEK(&HD
100+N*8+I)
1055 C(I)=PEEK(&HE100+N*8+I)
1060 FOR J=7 TO 0 STEP -1
1070 B(I,J)=P(I)-INT(P(I)/2)*2:
P(I)=INT(P(I)/2)
1080 IF B(I,J)=0 THEN LINE (96+
8*J+1,64+8*I+1)-(96+8*J+6,64+8*
I+7),C(I)AND15,BF  ELSE LINE (9
6+8*J+1,64+8*I+1)-(96+8*J+6,64+
8*I+7),(C(I)-(C(I)AND15))/16,BF
1090 NEXT J,I:RETURN
1430 FOR I=96 TO 160 STEP 8:LIN
E (I,64)-(I,128),1:NEXT
1440 FOR I=64 TO 128 STEP 8:LIN
E (96,I)-(160,I),1:NEXT
1450 RETURN
1500 N=0:GOSUB 9000:PRINT#1,"Us
e as setas para escolher":PRESE
```

```
80 ON A GOTO 90,130,100,130,110
,130,120,130
85 GOTO 130
90 IF Y>64 THEN Y=Y-8:GOTO 200
95 GOTO 200
100 IF X<152 THEN X=X+8:GOTO 20
0
105 GOTO 200
110 IF Y<120 THEN Y=Y+8:GOTO 20
0
115 GOTO 200
120 IF X>96 THEN X=X-8:GOTO 200
125 GOTO 200
130 IF K$="G" THEN GOSUB 1500:G
OTO 200
140 IF K$="R" THEN GOSUB 1000:G
OTO 200
150 IF K$="T" THEN GOSUB 2000:G
OTO 200
160 IF K$="C" THEN GOSUB 2500:G
OTO 200
170 IF K$="F" THEN GOSUB 2600:G
OTO 200
200 IF C+1>15 OR C+1=F THEN S=C
 ELSE S=C+1
210 PUT SPRITE 0,(X,Y),S
220 GOTO 60
500 SWAP C1,C2:LINE (X+1,Y+1)-(
X+6,Y+7),C2,BF
510 IF C2=F THEN B((Y-64)/8,(X-
96)/8)=0 ELSE B((Y-64)/8,(X-96)
/8)=1
520 C((Y-64)/8)=C*16+F
530 RETURN
1000 N=0:GOSUB 9000:PRINT#1,"Us
e as setas para escolher":PRESE
```

```
T(8,170):PRINT#1,"o número do b
loco :":CLOSE 1:GOSUB 9200
1510 X$=INKEY$:IF X$="" THEN 15
10
1515 IF X$=CHR$(13) THEN GOSUB
9100:GOTO 1550
1520 IF X$=CHR$(29) AND N>0 THE
N N=N-1:GOSUB 9200:GOTO 1510
1525 IF X$=CHR$(28) AND N<255 T
HEN N=N+1:GOSUB 9200:GOTO 1510
1530 IF X$=CHR$(31) AND N>9 THE
N N=N-10:GOSUB 9200:GOTO 1510
1540 IF X$=CHR$(30) AND N<245 T
HEN N=N+10:GOSUB 9200:GOTO 1510
1545 GOTO 1510
1550 FOR I=0 TO 7:P(I)=0:FOR J=
0 TO 7
1560 P(I)=P(I)+B(I,J)*2^(7-J)
1570 NEXT J
1580 POKE &HD100+N*8+I,P(I)
1585 POKE &HE100+N*8+I,C(I)
1590 VPOKE BASE(12)+N*8+I,P(I)
1600 VPOKE BASE(11)+N*8+I,C(I)
1610 NEXT I:RETURN
2000 GOSUB 9000:PRINT#1,"(S)AVE
 ou (L)OAD ?"
2010 X$=INKEY$:IF X$<>"S" AND X
$<>"L" THEN 2010
2020 GOSUB 9100:IF X$="L" THEN
2100
2030 GOSUB 9000:PRINT#1,"Posici
one o gravador":PRESET(8,170):P
RINT#1,"e aperte <ENTER>"
2035 IF INKEY$<>CHR$(13) THEN 2
035
2040 BSAVE "CAS:UDG",&HD100,&HF
37F
2050 GOSUB 9100:RETURN
2100 GOSUB 9000:PRINT#1,"Posici
one o gravador":PRESET(8,170):P
RINT#1,"e aperte <ENTER>"
2105 IF INKEY$<>CHR$(13) THEN ■
105
2110 BLOAD "CAS:"
2120 GOSUB 9100
2130 GOSUB 9000:PRINT#1,"(V) pa
ra ver o banco":PRESET(8,170):P
RINT#1,"(ENTER) para voltar à ■
dição"
2140 X$=INKEY$:IF X$="" THEN 21
40
2150 IF X$=CHR$(13) THEN GOSUB
9100:RETURN
2160 IF X$<>"V" THEN GOTO 2140
2170 FOR N=0 TO 255
2180 FOR J=0 TO 7
2190 VPOKE BASE(12)+N*8+J,PEEK(
&HD100+N*8+J)
2200 VPOKE BASE(11)+N*8+J,PEEK(
&HE100+N*8+J)
2220 NEXT J,N
2230 GOSUB 9100:RETURN
2500 GOSUB 9000:PRINT#1,"Número
 da cor de frente (0-F)"
2510 X$=INKEY$:IF X$="" THEN 25
10
2515 IF X$<"0" OR (X$>"9"AND X$
<"A") OR X$>"F" THEN 2510
2520 C=VAL("&H"+X$):C1=C:C2=F
2530 GOSUB 9100:RETURN
2600 GOSUB 9000:PRINT#1,"Número
 da cor de fundo (0-F)"
2610 X$=INKEY$:IF X$="" THEN 26
10
2615 IF X$<"0" OR (X$>"9"AND X$
<"A") OR X$>"F" THEN 2610
2620 F=VAL("&H"+X$):C1=C:C2=F
2630 GOSUB 9100:RETURN
9000 OPEN "GRP:" FOR OUTPUT AS
#1:PRESET(8,160):COLOR 1,15:RET
URN
9100 CLOSE 1:LINE (0,160)-(255,
191),15,BF:RETURN
9200 LINE (190,168)-(255,191),1
5,BF:GOSUB 9000:PRESET(190,168)
:PRINT#1,N:CLOSE1:RETURN
```

Até agora, sempre que precisávamos de um desenho em alta resolução, usávamos o comando **DRAW**. Este é muito bom para figuras móveis, devido à velocidade de operação. Ele apresenta, contudo, dois inconvenientes: ■ extensão dos dados, já que as regras de definição das figuras são bem complicadas, e ■ dificuldade na utilização de cores, pois as figuras ficam deformadas quando coloridas.

O Apple e ■ TK-2000 também podem mostrar na tela caracteres definidos pelo usuário, usando regras bem semelhantes às dos demais micros. Tais regras devem ser empregadas quando for importante ■ cor, mas não a velocidade. Convém, nesse caso, consultar as seções correspondentes, dedicadas ■ outros computadores.

Quando executamos o programa, ve■■ um quadriculado de ■ x 8 pontos na tela. Este será o "papel quadriculado" onde desenharemos nosso bloco gráfico.

Temos também um cursor, na forma de um ponto. Para movimentá-lo, utilizamos as teclas Z, X, P e L, como de costume. Usando ■ barra de espaço podemos "acender" o ponto correspondente, apagando-o com ■ tecla A.

Para obtermos caracteres coloridos, precisamos entender como ■ Apple e o TK-2000 representam cores ■■ tela. Embora nosso quadriculado tenha oito pontos de largura (ou seja, ■■ horizontal), na tela aparecerão apenas os sete primeiros (a partir da esquerda). São, portanto, quarenta posições com sete pontos cada, resultando nos 280 pontos da tela de alta resolução. Se um ponto da linha estiver apagado, aparecerá em preto. Caso esteja aceso, contudo, sua cor vai depender de sua posição *na tela*. Se o ponto ocupar uma posição par, será violeta. Se ocupar uma posição ímpar, será verde. Tais regras se aplicam quando o oitavo ponto, que fica na extrema direita do bloco e não aparece na tela, estiver apagado. Se o oitavo ponto estiver aceso, as cores violeta e verde serão substituídas por azul e vermelho, respectivamente. Se dois pontos adjacentes estiverem acesos, assumirão a cor branca.

Existem, assim, seis cores disponíveis no modo de alta resolução. Sua exibição, porém, está sujeita às seguintes limitações:

1. Ponto em coluna ímpar ■ preto, violeta ou azul.
2. Ponto em coluna par é preto, verde ou vermelho.
3. Cada linha de bloco pode ter ■■ cores violeta ■ verde ou azul ■ vermelha. Não é possível misturar cores de grupos diferentes na mesma linha.
4. Dois pontos adjacentes e acesos ficarão brancos, mesmo que estejam em linhas diferentes.

Deve-se ressaltar ainda que as cores dependem da posição do ponto *na tela, e não no bloco gráfico*. Isto quer dizer que ■ cor do ponto varia com ■ sua coordenada horizontal — de 0 a 279. Como a largura de cada linha de bloco na tela é de sete pontos, quando planejamos um bloco não sabemos quais os pontos que irão ocupar posições pares ou ímpares. Uma vez que tenhamos desenhado o nosso bloco, podemos guardá-lo no banco de memória do programa. Para isso, basta apertar a tecla G. O computador pergunta então em qual posição do banco vamos colocar ■ caractere — 0 ■ 300. Informado esse número, o caractere é mostrado na tela, em posição correspondente à do banco e em tamanho natural. Sua cor dependerá da posição no banco. Para facilitar o trabalho, é conveniente armazenar os blocos nas posições pares, onde é possível planejar as cores, pois as colunas da tela coincidem com as do bloco. Nas posições ímpares, as cores são de grupos invertidos, já que as posições pares e ímpares são invertidas.

O banco nada mais é que uma porção protegida de memória. Ele pode ocupar a página 2 de gráficos, tornando ■ trabalho mais simples. Essa página, porém, não deve ser usada enquanto o programa estiver no micro.

O uso da tecla R permite recuperar um bloco gravado ■■ banco, trazendo seu desenho para o quadriculado. O bloco recuperado, por sua vez, permanece ■■ banco. Um mesmo bloco pode ser guardado em diferentes posições.

COMO GRAVAR EM DISCO E FITA

Aqueles que dispõem de unidade de disco podem usar a tecla T para gravar ou carregar todo o conteúdo do banco

de memória. Feito isso, o computador perguntará qual a opção desejada. Se você selecionar a primeira opção, o conteúdo do banco será gravado. Não se esqueça, portanto, de guardar ■ último bloco editado.

Quando carregamos um banco, as posições que não contêm blocos são preenchidas por linhas brancas.

Quem não dispõe de disco, ou tem um TK-2000, deve parar o programa — com <**RESET**> ou <**CTRLX**><**C**> — para poder gravar ■ banco de memória em fita. Para tanto, é preciso usar o monitor, digitando **CALL -151** — ou **LM** no TK-2000. A seguir, digite (no Apple):

```
4000.5FFF W
```

No caso do TK-2000, temos algumas modificações:

```
A000.BFFF W "nome"
```

Para ler a fita, troque a letra **W** por **R**.

Depois de ter carregado o banco, execute novamente o programa. O conteúdo do banco poderá ser visualizado por meio da tecla T (caso tenham sido feitas as modificações sugeridas).

Na próxima parte do artigo, traremos mais informações sobre a criação ■ utilização de blocos gráficos no Apple. Além disso, novas funções serão acrescentadas ao gerador de blocos.

```
10  HOME : HGR : HCOLOR= 3: DIM
 B(8,8)
20 E = 16384:T = 8192
30  GOSUB 1030
50 X = 108:Y = 72: GOTO 200
60  GET K$:LX = X:LY = Y
70  IF K$ = "Z" AND X > 108 THE
N X = ■ - 8: GOTO 200
80  IF K$ = "X" AND X < 164 THE
N X = ■ + 8: GOTO 200
■  IF K$ = "P" ■ Y > 72 THEN
 Y = Y - 7: GOTO 200
100  IF K$ = "L" AND Y < 121 TH
EN Y = Y + 7: GOTO 200
110  IF K$ = " " THEN B((Y - 72
) / 7,(X - 108) / 8) = 1: HCOLO
R= 2: GOSUB 500: GOTO 200
120  IF K$ = "A" THEN B((Y - 72
) / 7,(X - 108) / 8) = 0: HCOLO
R= 0: GOSUB 500: GOTO 200
130  IF K$ = "G" THEN  GOSUB 15
00: GOTO 50
140  IF K$ = "R" THEN  GOSUB 10
00: GOTO 50
150  IF K$ = "T" THEN  GOSUB 20
00: GOTO 50
200  HCOLOR= 0: HPLOT LX + 3,LY
 + 4 TO LX + 4,LY + 3: HPLOT LX
 + 4,LY + 4 TO LX + 3,LY + 3
210  HCOLOR= 5: HPLOT X + 3,Y +
 4 TO X + 4,Y + 3: HPLOT X + 4,
Y + 4 TO ■ + 3,Y + 3
230  GOTO 60
500  FOR K = ■ + 1 TO X + 6: HP
LOT K,Y + 1 TO K,Y + 6: NEXT :
RETURN
1000  HOME : VTAB 22: INPUT "QU
AL O NUMERO DO BLOCO ?";N: IF N
 > 320 THEN 1000
1005  FOR I = 0 TO 7:P(I) =  PE
EK (E + ■ * 8 + I)
1010  FOR J = 0 TO 7
1015 B(I,J) = P(I) -  INT (P(I)
 / 2) ■ 2:P(I) =  INT (P(I) / 2
)
1020  HCOLOR= 2 * B(I,J): FOR ■
 = 109 + 8 * J TO 114 + 8 ■ J:
HPLOT K,73 + 7 * I TO K,78 + 7
* I: NEXT K,J,I
1025  HOME : RETURN
1030  HCOLOR= 3
1040  FOR I = 108 TO 172 STEP 8
: HPLOT I,72 TO I,127: NEXT
1050  FOR I = 72 TO 128 STEP 7:
 HPLOT 108,I TO 172,I: ■
1060  RETURN
1500  HOME : VTAB 22: INPUT "QU
AL O NUMERO DO BLOCO ?";N: IF ■
 > 320 THEN 1500
1502 L =  INT (N / 40):C = N -
L * 40
1505  FOR I = 0 TO 7:P(I) = 0:
FOR J = 0 TO 7
1510 P(I) = P(I) + B(I,J) * ■ ^
 J
1520  NEXT J: POKE E + N * 8 +
I,P(I)
1525  POKE T + L * 128 + C + 10
24 * I,P(I)
1530  NEXT I: HOME : RETURN
2000  HOME : VTAB 22: INPUT "(S
)AVE OU (L)OAD ?";A$: IF A$ <
> "S" AND A$ <  > "L" THEN 2000
2010  INPUT "NOME DO ARQUIVO ?"
;AR$
2020  IF A$ = "L" THEN 2100
2030  PRINT : PRINT  CHR$ (4);"
BSAVE ";AR$;",A";E;",L3000"
2040  ■ : RETURN
2100  PRINT : PRINT  CHR$ (4);"
BLOAD ";AR$
2110  FOR N = ■ TO 320
2120 L =  INT (N / 40):C = N -
L ■ 40
2130  FOR I = 0 TO 7
2140  POKE T + L ■ 128 + C + 10
24 ■ I, PEEK (E + N ■ ■ + I)
2150  NEXT I,N
2160  HOME : RETURN
```

Aqueles que não dispõem de unidade de disco não devem copiar as linhas 2000 a 2100. Uma instrução ■ deve ser colocada na linha 2000.

Os usuários do TK-2000 *não* devem copiar as linhas 2000 a 2100, fazendo ainda as seguintes modificações:

```
20 E = 40960:T = 8192
```

No Spectrum podem-se criar novos blocos...

Quando colocamos o programa para funcionar, ele nos faz duas perguntas iniciais, relacionadas com o tipo de bloco gráfico que queremos criar.

A primeira se refere ao **PMODE** desejado. **PMODE4** nos dá uma grande resolução em preto e branco ou em preto ■ verde apenas. A segunda opção, **PMODE3**, tem uma resolução três vezes mais baixa na horizontal — pontos com o triplo de largura —, mas permite que usemos quatro cores.

A segunda pergunta diz respeito ao conjunto de cores que vamos utilizar. **SCREEN0** no **PMODE4** nos dá preto e verde, enquanto em **PMODE3** nos dá vermelho, azul, verde e amarelo. **SCREEN1** nos dá preto ■ branco em **PMODE4** e ciano, magenta, laranja e branco em **PMODE3**.

Feitas estas opções, não poderemos mais mudar o **PMODE** sem usar **RUN** novamente, embora possamos mudar o **SCREEN**.

A seguir, ■ computador coloca na tela o quadriculado, que é ■ base para o desenho do bloco gráfico. Podemos então usar as setas para mover o cursor e assim desenhar ■ bloco.

Com o cursor sobre um determinado ponto, devemos pressionar <**ENTER**> para acender ■ ponto. Para apagar pontos, basta mudar sua cor, como veremos mais adiante.

Dessa forma, desenharemos nosso bloco rápida e facilmente. A porção seguinte do programa terá uma opção de uso de joystick.

Como podemos observar, o quadriculado não é o único quadrado na tela: abaixo dele há mais oito e à sua direita, mais dois.

Os dois da direita são versões atualizadas do bloco criado, em modo normal e invertido. À medida que desenhamos, esses dois "modelos" vão mudando, de maneira a possibilitar a visualização do resultado em tamanho real.

...ou [illegible] [illegible] caracteres do próprio micro.

Blocos mais largos [illegible] TRS-Color.

Os oito quadrados na parte inferior da tela servem p[illegible] guardar os blocos que já foram criados. Isso significa que até oito blocos podem ser guardados ao mesmo tempo no banco de memória do programa. Alguns desses blocos não são visíveis, uma vez que têm pontos com a mesma cor do pano de fundo. Eles não mudam junto com a edição, mas sim quando guardamos no banco um caractere recém-criado.

AS TECLAS DE CONTROLE

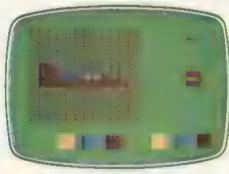
O programa tem uma série de teclas de controle, que nos possibilitam fazer opções.

Duas delas são fundamentais: as teclas S e G. S guarda no banco de memória o bloco que foi editado. Quando pressionamos essa tecla, o programa nos pede um número de 1 a 8, que corresponde à posição do bloco no banco. Isso permite que substituamos qualquer um dos blocos do banco.

Já [illegible] tecla G, seguida do número correspondente, faz [illegible] operação inversa — recupera um bloco do banco e o coloca no quadriculado. Embora o programa utilize uma rotina [illegible] linguagem de máquina, o processo leva alguns segundos.

Recuperar um bloco não modifica o conteúdo da posição correspondente no banco. Assim, podemos recuperar um determinado bloco do banco, trazendo-o para o quadriculado quantas vezes quisermos, desde que não guardemos nada na mesma posição do banco.

Essa parte do programa utiliza duas outras teclas de controle. Se pressionarmos C, mudaremos a cor. Se estivermos em um modo com quatro cores, podemos usar qualquer uma delas. No modo de duas cores, escolhemos [illegible] ou outra. Selecionada a cor adequada, podemos apagar qualquer ponto, corrigindo, assim, eventuais erros cometidos. Após digitar C, devemos informar o número da cor desejada.

A última tecla de controle dessa parte do programa [illegible] T, que permite [illegible] gravação ou a recuperação de todo um banco de memória contendo oito blocos.

```
10 CLEAR 200,30998:DEFUSR0=3100
0
20 T=0:FOR K=0 TO 43:READ N:T=T
+N:POKE K+31000,N:NEXT:READ C:I
[illegible] T<>C THEN PRINT "ERRO NOS DAD
OS":END
50 DATA 189,179,237,31,3,142,12
3,12,230,192,134,8,183,121,68,7
9,88,73,122,121,68,125,121,23
[illegible] DATA 39,5,88,73,122,121,68,1
67,128,125,121,68,38,233,140,12
5,76,38,221,57,4725
110 CLS:PRINT"SELECIONE MODO GR
AFICO (3-4)";
120 A$=INKEY$:IF A$<"3" OR A$>"
4" THEN 120
130 T=5-VAL(A$):PRINT A$
140 PRINT"SELECIONE TIPO DE TEL
A(0-1)";
150 A$=INKEY$:IF A$<"0" OR A$>"
1" THEN 150
160 PRINT A$:ST=VAL(A$)
200 PMODE 5-T,1
210 DIM A(14),C1(1),C2(1),C3(1)
,C4(1)
220 PCLS 4:GET(0,0)-(9,5),C4,G
230 PCLS 3:GET(0,0)-(9,5),C3,G
240 PCLS 2:GET(0,0)-(9,5),C2,G
250 PCLS 1:GET(0,0)-(9,5),C1,G
260 PCLS:SCREEN 1,ST
270 C=-1:F=3*T-2:GOSUB 2070
280 FOR X=8 TO 255 STEP 32:C=(C
+1) AND 3:COLOR C+1:LINE(X,165)
-(X+23,188),PSET,BF:NEXT:COLOR
6-T
290 FOR X=3 TO 147 STEP T*6:LIN
E (X,3)-(X,147),PSET:NEXT
300 FOR Y=3 TO 147 STEP 6:LINE
(3,Y)-(147,Y),PSET:NEXT
310 X=12:Y=12
320 X1=X*6+4:Y1=Y*6+4
330 PUT(X1,Y1)-(X1+5*T-1,Y1+4),
C1,NOT
340 A$=INKEY$
350 GOSUB 1500
360 IF A$="" THEN 380
370 ON INSTR("PCTRGSMIV",A$) GO
SUB 2200,2500,2300,2800,2600,27
00,2900,2100,2400
380 IF Y<0 THEN Y=23
390 IF Y>23 THEN Y=0
400 IF X<0 [illegible] X=24-T
410 IF X>23 THEN X=0
420 GOTO 320
1500 PUT (X1,Y1)-(X1+5*T-1,Y1+4
),C1,NOT
1510 IF PEEK(338)=191 GOSUB2000
1520 IF PEEK(343)=247 THENX=X-T
1530 IF PEEK(344)=247 THENX=X+T
1540 IF PEEK(341)=247 THENY=Y-1
1550 IF PEEK(342)=247 THENY=Y+1
1560 RETURN
2000 GOSUB 4000
2010 P=3*Y+INT(X/8):PX=7-X+8*IN
T(X/8)
2020 IF T=2 THEN VL=F-1 ELSE VL
=-(F=1)
2030 PK=PEEK(P+VARPTR(A(0)))
2040 PK=PK AND(255.1-2^PX):IF T
=2 THEN PK=PK AND(255.1-2^(PX-1
))
2050 PK=PK OR VL*2^(PX+1-T)
2060 POKE P+VARPTR(A(0)),PK
2070 PUT (216,10)-(239,33),A,PS
ET
2080 PUT (216,70)-(239,93),A,PR
ESET
2090 RETURN
2100 RETURN
2200 RETURN
2300 A$=INKEY$:IF A$<>"S" AND A
$<>"L" THEN 2300
2310 IF A$="S" THEN 2330
2320 CLOADM:SCREEN 1,ST:RETURN
2330 CSAVEM"",6800,7679,6800
2340 SCREEN 1,ST:RETURN
2400 ST=1-ST:SCREEN 1,ST:RETURN
2500 A$=INKEY$:IF A$<"0" [illegible] A$>
"8" THEN 2500
2510 F=((VAL(A$)-1)AND 3)+1:RET
URN
2600 A$=INKEY$:IF A$<"1" OR A$>
"8" THEN 2600
2610 J=VAL(A$)-1
2620 GET(J*32+8,165)-(J*32+31,1
88),A
2630 GOSUB 3000:GOTO 2070
2700 A$=INKEY$:IF A$<"1" [illegible] A$>
"8" THEN 2700
2710 J=VAL(A$)-1
2720 PUT(J*32+8,165)-(J*32+31,1
88),A,PSET
2730 RETURN
2800 RETURN
2900 RETURN
3000 CL=F:POKE 30999,T-1:N=USR0
(VARPTR(A(0)))
3010 FOR K=0 TO 23:FOR J=0 TO 2
3 STEP T: F=PEEK(31500+K*24/T+J
/T)+3-T
3020 X1=J*6+4: Y1=K*6+4
3030 GOSUB 4000:NEXT J,K:F=CL:R
ETURN
4000 [illegible] F GOTO 4010,4020,4030,4
040
4010 PUT(X1,Y1)-(X1+5*T-1,Y1+4)
,C1,PSET:RETURN
4020 PUT(X1,Y1)-(X1+5*T-1,Y1+4)
,C2,PSET:RETURN
4030 PUT(X1,Y1)-(X1+5*T-1,Y1+4)
,C3,PSET:RETURN
4040 PUT(X1,Y1)-(X1+5*T-1,Y1+4)
,C4,PSET:RETURN
```

```
10 HOME
20 LET K=PEEK(39)
30 IF K<>48 THEN PRINT K
40 GOTO 20
```

A linha 20 copia o conteúdo da memória 39 na variável **K**. A 30 mostrará esse valor na tela apenas se ele for diferente de 48, ou seja, se alguma tecla tiver sido pressionada. Em caso contrário, o programa saltará de novo para a linha 20, de modo a manter vigilância sobre a memória 39.

O único problema é que os códigos numéricos retornados para cada tecla pressionada não correspondem ao código ASCII dos caracteres aos quais elas estão atribuídas.

Essa dificuldade, entretanto, pode ser facilmente solucionada por uma pequena tabela de conversão, mostrada no fim deste artigo. Mas, se você preferir, pode descobrir quais são esses códigos usando o programa.

Para entender melhor como o comando **PEEK** (39) pode ser usado no lugar do comando **INKEY$**, acompanhe o programa apresentado a seguir. Ele desenha uma reta na tela, orientada numa determinada direção. Pressione uma das teclas de controle do cursor (flechas) uma única vez, e a linha sofrerá uma mudança de direção.

Ao ser acionada a tecla **<FIRE>**, o programa terminará.

```
10 HGR2:HCOLOR=3
20 LET X=100:Y=90:DX=1:DY=0
30 HPLOT X,Y
40 FOR I=1 TO 40:NEXT I
50 LET P=PEEK(39)
60 IF P=48 THEN X=X+DX:Y=Y+DY:
   GOTO 30
70 IF P=36 THEN DX=0:DY=-1:GOTO
   30
80 IF P=30 THEN DX=0:DY=1:GOTO
   30
90 IF P=18 THEN DX=-1:DY=0:GOTO
   30
95 IF P=24 THEN DX=1:DY=0:GOTO
   30
100 IF P=46 THEN TEXT:END
110 GOTO 30
```

O laço principal do sistema está compreendido entre as linhas 30 e 110. A cor e a posição inicial do ponto da tela são definidos nas linhas 10 e 20. A linha 20 também define duas variáveis, **DX** e **DY**, que darão o *incremento* a ser somado a cada coordenada — X e Y —, quando o ponto for deslocado. Note que, com **DX = 1** e **DY = 0**, inicialmente o ponto vai se deslocar de um em um só na direção horizontal.

A linha 50 examina a memória vinculada ao teclado. Se nenhuma tecla tiver sido pressionada, o código resultante é 48, e as variáveis **X** e **Y** são incrementadas de **DX** e **DY**, respectivamente, na linha 60.

As linhas 70 a 100 verificam qual foi a tecla pressionada. Observe que os códigos 36, 30, 18 e 24 correspondem às teclas do cursor; se uma destas for pressionada, os valores de **DX** e **DY** serão ajustados de modo a provocar movimento na direção apontada (a linha 40, por sua vez, serve apenas para diminuir a velocidade de deslocamento do ponto na tela).

Finalmente, a linha 100 interromperá o programa, quando a tecla **<FIRE>** for pressionada.

UM INKEY$ MAIS SOFISTICADO

O uso do comando **PEEK** implica a necessidade de trabalhar com valores não padronizados de códigos de teclas. Isso pode ser desvantajoso em muitas aplicações onde se deseja receber o código ASCII da tecla, como também em programas de senha secreta (veja página 166), e outros.

Infelizmente, não existe nenhum **PEEK** apropriado para trabalhar com valores ASCII. Mas podemos "enxertar" em nosso programa em BASIC uma pequena rotina em linguagem de máquina (cujo funcionamento não será explicado por enquanto), que atribua a um **PEEK** essa propriedade.

Digite o comando **NEW** para apagar o programa anterior, e entre e execute o programa abaixo:

```
1 FOR I=768 TO 774
2 READ N:POKE I,N:NEXT I
3 DATA 32,67,240,141,0,4,96
```

Como você deve ter notado, aparentemente não acontece nada. Tudo que o programa faz é carregar um programinha de sete bytes em código de máquina numa parte não usada da memória do micro, que começa na locação 768. Isso é feito por intermédio do comando **POKE** na linha 2. O programa está armazenado na linha **DATA**.

Depois de rodar o programa, você pode apagá-lo com o comando **NEW**, pois ele não será mais necessário (o programa em código de máquina ficará armazenado até que a máquina seja desligada). Mas, se você preferir, pode também deixá-lo onde está.

Para fazer funcionar esse programa, é preciso utilizar o comando **CALL 768** (que indica o endereço onde começa o programa). Toda vez que isso é feito, o computador "varre" o teclado e retorna a um valor numérico correspondente ao código ASCII da tecla pressionada, mais 128. Se nenhuma tecla for pressionada, o código 0 será retornado. Para encontrar esse valor, usamos o comando **PEEK (1024)**.

O programa abaixo faz um teste.

```
10 HOME
20 CALL 768
30 K=PEEK(1024)
40 IF K>0 THEN PRINT K
50 GOTO 20
```

Como você já deve ter observado, a lógica do programa é bem clara: a linha 20 chama a rotina de máquina e retorna o código da tecla pressionada na locação 1024 de memória.

A linha 40 imprimirá esse valor se este for maior do que 0.

Ao executar o programa você notará que os valores retornados aparentemente não pertencem ao código ASCII. A letra A maiúscula, por exemplo, retorna 193; a letra B, 194, e assim por diante. Experimente diminuir 128 desse valor: o resultado será 65, 66 etc.: ou seja, o código ASCII.

Tente agora substituir a linha 40 pelo seguinte programa:

```
40 IF K>0 THEN PRINT CHR$(K);
```

Em seguida:

```
40 IF K>159 THEN PRINT CHR$(
   K-128);
```

Qual a diferença entre estas duas versões?

UM PEQUENO TESTE

Agora, como exemplo do uso do comando **CALL**, digite o programa:

```
10 HOME:SPEED=100
20 PRINT "TESTE DE DIGITACAO":
   PRINT
30 FOR I=1 TO 10
40 LET T=0:N$=""
50 FOR N=1 TO 7
55 LET R=48+INT(RND(1)*10)
60 N$=N$+CHR$(R):NEXT N
70 VTAB 10:HTAB 15:PRINT N$
80 VTAB 20:HTAB 1:PRINT "
   ":VTAB 20:HTAB 1
90 LET R$=""
100 FOR J=1 TO 7
110 CALL 768:K=PEEK(1024)
115 IF K>0 THEN 130
120 LET T=T+1:IF T>200 THEN 180
125 GOTO 110
130 LET K$=CHR$(K-128):R$=R$+K$
140 PRINT K$;
150 NEXT J
155 VTAB 10:HTAB 15
160 IF R$=N$ THEN PRINT
    "ACERTOU":GOTO 190
170 PRINT "ERROU !":GOTO 190
180 VTAB 10:HTAB 15:PRINT
    "DEMOROU"
```

```
190 FOR J=1 TO 1000:NEXT J
200 NEXT I
```

O programa gera uma seqüência aleatória de sete dígitos [illegible] pede que eles sejam teclados rapidamente. O objetivo do exercício [illegible] digitar o número todo, [illegible] errar, dentro de um intervalo de tempo máximo.

As linhas 50 a 70 geram esse número, armazenando-o na variável **N$** e mostrando-o na tela. As linhas 100 a 150 efetuam a leitura das teclas pressionadas pelo usuário e a colocam em uma variável **R$**.

Finalmente, as linhas 155 a 190 verificam qual foi [illegible] resultado — este pode indicar várias alternativas, como erro, acerto, ou demora em datilografar todo o número de teste.

As linhas 110 [illegible] 115 efetuam a "varredura" do teclado para verificar se algo foi pressionado. Em seguida, [illegible] linha 120 aumenta [illegible] contagem de tempo (**T**), enquanto [illegible] usuário não pressiona nenhuma tecla. Se alguma tecla foi pressionada, as linhas 130 e 140 concatenam [illegible] resultado na variável de resposta e mostram o caractere em outra locação da tela.

COMO FUNCIONA

Quem conhece um mínimo de código de máquina será capaz de entender facilmente o funcionamento da rotina 768. Você pode tentar carregá-la no TK-2000 usando [illegible] Míni-Assembler, em lugar do programa apresentado linhas atrás. Simplesmente digite **ASS** e pressione <**RETURN**>. Depois, cada vez que aparecer um sinal de exclamação, digite as linhas abaixo.

Para sair, pressione as duas teclas <**RESET**> ao [illegible] tempo:

```
300: JSR $F043
303: STA $400
306: RTS
```

O número 300 está em hexadecimal: equivale ao 768 em decimal e [illegible] o endereço de origem da rotina (mas poderia ser um número diferente deste).

O comando **JSR** pula para [illegible] subrotina em código de máquina armazenada no endereço FO43, hexadecimal, que efetua a varredura do teclado (**JSR** equivale ao **GOSUB** do BASIC).

O comando **STA** copia o conteúdo do acumulador do micro (onde está o código da tecla pressionada) e o armazena na memória 1024 ($400 em hexa; mas poderia ser outra).

Finalmente, o comando **RTS** retor[illegible] para o BASIC.

TABELA DE CÓDIGOS DE TECLA

Caractere	ASCII	PEEK (39)	CALL 768	Caractere	ASCII	PEEK (39)	CALL 768
nenhum	-	48	0	C	67	3	195
espaço	32	12	160	[illegible]	68	[illegible]	196
!	33	151	161	E	69	15	197
"	34	150	162	F	70	8	198
[illegible]	35	149	163	G	71	7	199
[illegible]	36	148	164	H	72	37	200
[illegible]	37	147	165	I	73	33	201
&	[illegible]	153	166	J	74	[illegible]	202
'	39	154	167	[illegible]	75	39	203
(	40	155	168	L	76	40	204
)	41	156	169	M	77	44	205
*	42	157	170	[illegible]	78	43	206
+	43	163	171	O	79	34	207
,	44	45	172	P	[illegible]	35	208
-	45	161	173	Q	81	17	209
.	46	46	174	R	82	14	210
/	47	175	175	S	83	10	211
0	48	29	176	U	84	32	212
[illegible]	49	23	177	V	85	2	213
2	50	22	178	W	86	16	214
3	51	21	179	X	87	4	215
4	52	20	180	Y	[illegible]	31	216
5	53	19	181	Z	89	5	217
6	54	25	182	↑	112	36	240
7	55	26	183	↓	113	30	241
8	56	27	184	←	8	18	136
9	57	28	185	→	21	24	149
A	65	11	193	<FIRE>	46	46	174
[illegible]	66	1	194	<RETURN>	13	42	141

COMO FUNCIONA O PRINT USING

- O QUE É UM **PRINT** FORMATADO
- PARA QUE SERVE O **PRINT USING**
- COMO USAR MÁSCARAS DE FORMATAÇÃO

O comando PRINT USING controla ■ forma com ■ qual aparecem na tela os números ■ ■ cadeias alfanuméricas.

Embora recursos de formatação como ■ vírgula, ■ função **TAB** etc., funcionem bem na maioria das aplicações, em alguns casos é interessante controlar não só o posicionamento de números ■ nomes a serem exibidos no vídeo, como também os seus formatos. Ora, isso não pode ser feito apenas pelo comando **PRINT**. Alguns micros dispõem para isso de um poderoso comando auxiliar do **PRINT**, que é chamado **USING**. Outros, porém, não contam com ■ recurso. É ■ caso das linhas ZX-81, Spectrum, Apple II e TK-2000. Mais adiante, estudaremos um modo de superar essa dificuldade, obtendo efeitos semelhantes aos provocados pelo comando **PRINT USING**.

PARA QUE SERVE O PRINT USING

O **PRINT USING** facilita a especificação de formatos de saída em tela, possibilitando ■ programação de máscaras de saída. Para entender melhor como o **USING** funciona, digite o exemplo a seguir, que mostra as raízes quadradas de dez números inteiros:

```
10 CLS
20 FOR I=1 TO 10
30 PRINT USING "##    ##.####";
I;SQR(I)
40 NEXT I
```

Os números ficam alinhados em duas colunas e a coluna da direita tem todos os valores com quatro decimais. Para verificar o efeito do **PRINT USING**, substitua ■ linha 30 por:

```
30 PRINT I,SQR(I)
```

Para usar a especificação **USING** dentro de um **PRINT** basta colocar entre aspas o gabarito de saída. O caractere sustenido (#) indica onde deve ir cada dígito dos elementos da lista de saída. Um espaço em branco assinala onde começa e termina o gabarito relativo ■ cada elemento. O ponto indica onde deve ser feita a quebra entre as partes inteira e fracionária.

O gabarito estabelecido pelo **PRINT** pode ter especificações sucessivas para mais de um item de saída (no exemplo, o mesmo **PRINT** mostra a variável **I** e o resultado da raiz quadrada de **I**, sendo o primeiro com três dígitos). Enquanto existirem elementos de saída a serem impressos, *após* o **USING**, o computador continuará procurando gabaritos dentro da cadeia entre aspas.

O **USING** aceita ainda variáveis alfanuméricas.

A função **STRINGS$** gera uma cadeia de caracteres, com o comprimento desejado:

```
10 CLS
12 INPUT "QUANTAS DECIMAIS ";N
15 U$="##    ##." + STRINGS(ND ,
"#")
20 FOR I=1 TO 10
30 PRINT USING U$;I;SQR(I)
40 NEXT I
```

A tabela apresentada nesta página relaciona os diversos sinais convencionais utilizados em um **PRINT USING** e seus efeitos para ■ especificação de gabaritos, para números etc.

O **PRINT USING** encontra ■ aplicações mais interessantes na formatação de relatórios de saída em tela ou impressora, onde se requer um controle perfeito sobre o alinhamento de colunas, a formatação de valores numéricos etc. Uma única variável ou especificação entre aspas no **USING** pode ser usada para formatar uma linha de saída inteira da tabela. Se ■ linha for muito longa, o programa deve ser estruturado com base em vários **PRINT USING** separados por pontos ■ vírgulas.

É comum, em tabelas com valores monetários, empregar-se os sinais + e − no final (e não no começo) da quantia. Isso também é possível no **PRINT USING**.

Em programas para preenchimento automático de cheques, os sinais $ (cifrão), $$ (duplo cifrão), ** (protegido) e **$ (cifrão protegido) especificam como será preenchida ■ parte esquerda de campos com valores monetários:

```
10 M$=" **$###.###.###.##
cruzados"
15 D$=" ##/##/####"
20 INPUT "QUANTIA A SER PAGA "
; Q
30 INPUT "DIA,MES,ANO";D,M,A
40 CLS
50 PRINT "PAGUE-SE A QUANTIA DE
" ;
60 PRINT USING M$;Q;
70 PRINT "NO DIA ";:PRINT USING
D$;D;M;A
```

O TRS-80 e o TRS-Color têm as mesmas regras de uso do **PRINT USING**. Os sinais demarcadores são os mesmos, e a palavra **USING** deve suceder a expressão **PRINT**. Já ■ MSX apresenta algumas diferenças:

- Em vez do sinal de porcentagem (%), esse micro utiliza uma barra inversa (\) para demarcar o espaço das cadeias alfanuméricas.
- O ampersand (&) serve para indicar uma substituição completa por ■ cadeia alfanumérica.
- O termo **USING** pode ser misturado aos itens de uma linha de impressão:

```
70 PRINT "NO DIA ";USING D$;D;
M;A
```

CARACTERES DO PRINT USING

Sinal	Efeito
#	coloca um dígito numérico
.	coloca um ponto decimal
,	separa a parte inteira em grupos de três
−	coloca um sinal negativo após um número negativo, ou branco após um positivo
+	coloca um sinal − após um número negativo ou um sinal + após um positivo
^	assinala um formato exponencial
**	preenche um campo com asteriscos à esquerda
$$	coloca um cifrão antes de um valor numérico
**$	coloca um cifrão e asteriscos à esquerda
!	coloca o primeiro caractere de uma cadeia
%	assinala o início e o fim de um campo alfanumérico

APERFEIÇOE SUAS TELAS

- A IMPORTÂNCIA DE PLANEJAR
- COMO POSICIONAR O TEXTO
- MELHORE A DIAGRAMAÇÃO
- ADICIONE COR

Para dar a ■ programas uma aparência profissional, você precisará planejar cuidadosamente a página-título ■ outras telas. Aqui você aprende a fazer isso.

Do ponto de vista do usuário, há várias diferenças importantes entre um programa simples e bem planejado e outro que não apresenta essas qualidades. A primeira coisa que distingue um programa "profissional" de um amadorístico é, sem dúvida, o funcionamento adequado, sem erros e sem rotinas inúteis ou mal aproveitadas. Além disso, num nível mais teórico, o programa propriamente dito deve ser bem estruturado e claro. As técnicas necessárias para lhe assegurar uma boa estruturação e ■ ausência de erros já foram vistas em artigos anteriores.

Porém, mesmo o programa mais bem-feito parecerá obra de um principiante se não tiver ■ boa apresentação e, sobretudo, se as telas não forem bem organizadas ■ fáceis de compreender. Para isso, cada tela, bem como ■ menu ou mensagem que contiver, deve ser cuidadosamente estudada. Sem ■ posicionamento correto das declarações PRINT e INPUT será impossível criar telas de apresentação clara ■ interessante.

No artigo da página 146, examinamos os vários comandos BASIC disponíveis no seu computador que servem para controlar a posição dos caracteres na tela. Agora, você verá como usá-los para montar uma página-título para um jogo imaginário — "INPUT". Técnicas similares poderão ser empregadas para a elaboração de telas que apresentam instruções, menus e mensagens.

INSTRUÇÕES CLARAS

Nada causa pior impressão do que uma tela contendo mensagens com erros de grafia ou sintaxe.

Você já deve ter observado em muitos jogos mensagens do tipo "Você tem um canhão disponível". Este caso particular desabona ■ programador, sobretudo porque o erro poderia ter sido facilmente corrigido com uma declaração **IF...THEN** (IF L=1 THEN PRINT "CANHÃO DISPONÍVEL"). Antes de começar ■ se preocupar com os aspectos visuais da tela, procure assegurar-se de ter eliminado todos os erros desse tipo de programa.

Ao diagramar uma tela, leve em conta algumas regras básicas. Em primeiro lugar, não perca de vista o objetivo principal: a clareza. Palavras muito próximas ■ das outras dificultam a leitu■ — assim, deixe um espaço razoável entre as linhas. Evite também dividir palavras; se for impossível, utilize, então, o hífen para separar ■ duas partes.

Caso precise imprimir ■ tela dados fornecidos pelo usuário — como um nome, em uma lista de recordes —, tenha o cuidado de evitar que ■ novo dado interfira nos demais. No exemplo citado, você poderia incluir uma rotina que não aceitasse nomes com mais de um certo número de caracteres ou que, independentemente do tamanho do novo dado, o colocasse numa posição que não afetasse a do placar ou de outras informações exibidas.

Sempre que você tiver uma lista de informações, faça com que todos os dados comecem numa mesma coluna. A recomendação parece óbvia, mas vários programadores não procedem assim.

O segundo aspecto a observar diz respeito ■ quantidade de informações. Se você incorrer no erro de introduzir informações em excesso numa mesma tela, quem utilizar seu programa não conseguirá memorizar todas ■ instruções e, em conseqüência, não explorará todas as suas possibilidades.

Por outro lado, para não ser obrigado ■ trabalhar com um número exagerado de telas, você não poderá reduzir muito a quantidade de informação em cada uma delas. O ponto de equilíbrio dependerá, naturalmente, do seu programa — quanto maior sua facilidade de uso e clareza, menos instruções serão necessárias.

O USO DE ■

Torne a tela o mais interessante possível. Se puder use cores. Elas dão mais vida à tela, facilitando a compreensão das mensagens ou ressaltando determi-

nados itens. Estude cada ■ para decidir se utilizará cores no fundo, no texto, em ambos ou simplesmente na composição de uma moldura.

Os usuários do Apple ou do Sinclair Spectrum dispõem de um comando muito versátil — o **FLASH**. Ele apresentará sua mensagem piscando na tela (em cores, no Spectrum). Impedindo que as instruções assumam uma forma muito estática, esse recurso torna o texto bastante atraente.

Se você possui outro modelo de micro, poderá obter um efeito similar ao do comando **FLASH**, imprimindo o texto repetidamente no mesmo ponto da tela, em cores diferentes.

O comando **INVERSE**, do Apple, também possibilita o destaque de mensagens. Ele é especialmente útil, já que não se pode usar cor na tela de texto desse computador.

POSICIONAMENTO

Para assegurar a clareza da tela, falta ainda um importante elemento: o posicionamento das palavras. Para ter um exemplo, digite e execute este pequeno programa. Você verá *como não fazer* uma página-título para o seu programa. As palavras estão jogadas na tela: não há coordenação entre uma e outra, o que resulta numa grande confusão visual. Mais tarde, examinaremos como arrumar tudo isso.

```
10 PRINT "Apresentamos um nov
■ jogo        chamado input"
20 PRINT AT 5,17;"(c) 1986"
30 PRINT AT 12,13;" por Nova
Cultural"
40 PRINT AT 18,18;"qualquer t
ecla"
50 PAUSE 100
60 CLS : STOP
```

```
10 CLS 4
20 PRINT @40,"INPUT"
30 PRINT @136,"copyright nova c
ultural";
40 PRINT @228,"APRESENTA"
50 FOR T=1 TO 2000:NEXT:CLS
```

Para o TRS-80, use apenas **CLS** na linha 10 e multiplique por 2 todos os valores das instruções **PRINT@**.

```
10 PRINTTAB(4);"novacultural ap
resenta"
20 PRINTTAB(10);"input"
30 PRINT:PRINT"copyright 1986"
40 FORJ=1TO500:NEXT
50 CLS
```

```
10  PRINT  TAB( 5);"NOVACULTURA
L APRESENTA"
20  PRINT  TAB( 22);"INPUT"
30  PRINT : PRINT "COPYRIGHT 19
86"
40  FOR J = 1 TO 1000: NEXT
50  HOME
```

Observe o resultado: trata-se, sem dúvida, de uma tela inexpressiva, em que vários detalhes precisam ser alterados. Inicialmente, poderíamos empregar letras maiúsculas para melhorar ■ efeito. A versão para o Spectrum utiliza letras minúsculas até mesmo para a primeira palavra da tela e para o nome do jogo. Maiúsculas ressaltam palavras importantes ou, se forem usadas no texto inteiro, garantem ■ tela uma aparência bem clara.

Deveríamos, também, limpar a tela, antes de mostrar um novo conjunto de mensagens. Como você pôde observar no programa anterior, fragmentos do que estava na tela misturam-se à mensagem, completando a confusão.

Não há espaço entre as palavras NOVA e CULTURAL. Deveria haver. Veremos, adiante, como deixar tudo isso em ordem.

Outra falha do programa é não esperar que se pressione alguma tecla para prosseguir. Assim, não há muito tempo para ler as mensagens antes que elas sejam apagadas. Para evitar esse problema é sempre interessante incluir uma linha dizendo o que se deve fazer para que o programa continue — quando o usuário quiser.

No Spectrum, sabendo-se o tamanho da tela e o número de caracteres que compõem as palavras, pode-se planejar com muita facilidade o posicionamento da mensagem.

Suponhamos que seu computador tenha 22 linhas e 32 colunas; você quer colocar uma frase de dez caracteres na segunda linha, e bem no meio dela. Ao verificar o tamanho da frase, não se esqueça dos espaços.

Para calcular a coordenada horizontal da posição onde a frase começará a ser impressa, subtraia o tamanho da frase (10 no nosso caso) do total de caracteres por linha (32) e divida o resultado por 2.

No nosso exemplo, a resposta é 11. Como o 0 é considerado, o primeiro caractere da linha tem coordenada 0, e não 1. Assim, devemos subtrair 1 do resultado acima para conseguir a coordenada horizontal do **PRINT AT** (ou **PRINT TAB**).

Se você quiser posicionar uma frase a apenas dois ou três caracteres da margem esquerda, será mais fácil colocar o número de espaços dentro da declaração **PRINT** em vez de usar **PRINT TAB** ou **PRINT AT**. No entanto, se o número de espaços for grande, evite essa técnica, pois ela tende a gastar muita memória.

Você pode calcular ■ posição vertical como calcula a horizontal — ou seja, para colocar ■ mensagem no centro da tela, vê quantas linhas ela ocupa, subtrai esse número do total de linhas e divide por 2. Por fim, subtrai 1 do resultado, considerando que a coordenada da primeira linha é 0.

Quando quiser colocar ■ linha em outra posição, use um papel gráfico para visualizar melhor a tela, ou, se já tem alguma experiência, estime a posição que lhe convém, de memória.

Os usuários do Spectrum devem observar que a instrução **PRINT AT** funciona de modo um pouco diferente do dos outros computadores. O primeiro número que acompanha essa instrução refere-se à coordenada vertical (y) e ■ segundo, à coordenada horizontal (x). No entanto, o comando **PLOT** utiliza primeiro ■ coordenada x ■ depois a y.

O número após o **PRINT@** refere-se a uma posição da tela de texto do computador. Existem 512 posições no TRS-Color ■ 1024 no TRS-80. Os números, assim, não podem exceder 511 e 1023, respectivamente, já que ■ numeração começa no 0.

O cálculo do número da posição que se quer ■ muito simples. Primeiro, multiplique o número da linha desejada por 32 (esse é o número de caracteres por linha, no Color) ou por 64 (se você está usando um TRS-80). Depois, adicione um valor de 0 a 31 (no Color) ou de 0 a 63 (no TRS-80), para obter ■ posição horizontal.

Se preferir, deixe que o computador faça as contas por você. Para isso, utilize a seguinte fórmula: **PRINT@ L*32 + C, "MENSAGEM"**. L representa a linha que você deseja (de 0 a 14) e C a coluna (de 0 ■ 31). No TRS-80, use

64 em lugar de 32; C pode variar de 0 ■ 63.

Para centralizar uma frase na tela, subtraia o total de caracteres da frase do total de caracteres da linha (32 ou 64); divida o resultado por 2 e subtraia 1. Você obtém, assim, o número da coluna em que a impressão da frase ■ iniciará. Esse valor pode ser utilizado, por exemplo, com a fórmula dada anteriormente. Lembre-se de que os números que seguem ■ instrução **PRINT@** devem ser inteiros.

MSX

O posicionamento de mensagens no MSX também ■ muito fácil. Esse computador usa ■ comando **LOCATE**, seguido da coluna ■ linha (nesta ordem). Assim, para colocar ■ mensagem no lugar desejado, basta calcular a coluna e a linha. O mesmo comando também pode ser usado com um argumento numérico, que será interpretado como ■ coluna onde se iniciará a impressão.

Como o MSX tem duas telas de texto, verifique qual delas está em uso para, então, calcular a posição adequada de impressão. A primeira (**SCREEN0**) apresenta quarenta colunas por 24 linhas ■ a segunda (**SCREEN1**), 32 colunas por 24 linhas.

Para centralizar uma frase em determinada linha, subtraia ■ total de caracteres da frase do total de caracteres da linha e divida ■ resultado por 2. Esta será a posição para o início da frase.

O Apple e seus compatíveis empregam, além do **PRINTTAB**, os comandos **HTAB** ■ **VTAB** para posicionar o cursor dentro da janela de texto. O comando **HTAB** determina a posição horizontal com valores que variam de 1 a 40 e o **VTAB** determina ■ posição vertical, com valores que vão de 1 a 24. O ■ de valores fora dessas faixas provocará o aparecimento de uma mensagem de erro.

Para a centralização de uma mensagem, primeiro calcule o total de caracteres dela, depois subtraia de 40 e divida por 2. O resultado obtido deve ser utilizado como posição inicial de impressão da mensagem.

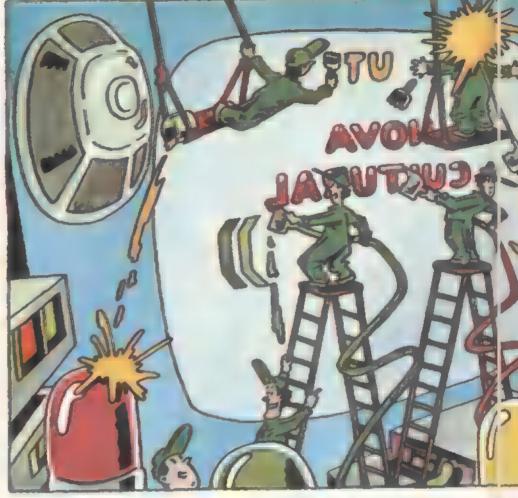
O interessante, nesse computador, é a facilidade com que se pode manipular ■ janela de texto. Essa característica torna muito simples ■ limpeza da linha de uma determinada posição até o fim dela ou de uma posição até o fim da tela. Utiliza-se ■ comando **CALL** ■ para apagar tudo ■ que estiver à direita da posição do cursor até o fim da linha. O **CALL -958** limpa todos os caracteres da janela de texto, da posição do cursor até o fim da tela. Desse modo, podemos trocar mensagens em locais específicos da tela, conservando ■ resto intocado, com grande rapidez e versatilidade.

UMA TELA BEM-FEITA

Uma tela pode ser clara ■ atraente mesmo que não empreguemos técnicas de programação complicadas. O programa dado a seguir é um exemplo disso. Ele corrige os defeitos do programa anterior ■ adiciona cor e animação ao título, para torná-lo mais interessante. Dê especial atenção ao uso dos comandos de controle do cursor, pois eles são fundamentais em tudo o que diz respeito à impressão na tela.

```
■ LET X=1
10 BORDER 1: PAPER 1: INK 7:
CLS
20 PRINT  INVERSE 1;AT 3,4;"
EDITORA NOVA CULTURAL "
30 PRINT  INVERSE 1;AT 5,10;"
 APRESENTA "
40 PAUSE 50
50 PRINT  PAPER 6; INK 1;AT
10,10;" I N P U T "
60 PRINT  PAPER 6; INK 2;FLASH
1;AT 9,9;"■■■■■■■■■■■■■
";AT11,9;"■■■■■■■■■■■■■
70 PRINT  PAPER 6; INK 2;
FLASH 1;AT 10,9:"■";AT 10,21;
"■"
80 PRINT  PAPER 5; INK 0;AT
15,2;"COPYRIGHT AUDIO.VISUAL,
1984"
90 PRINT ''''" QUALQUER TECLA
 PARA CONTINUAR"
100 PAUSE 0
110 CLS
120 PRINT "INPUT - INDICE"
130 PRINT
140 FOR X=1 TO 10
150 READ a$
155 READ b$
160 PRINT 'TAB 1;a$;TAB 25;b$
170 NEXT ■
180 DATA "Animacao","26-32","B
asic,programacao","2-7","BREAK
,Spectrum","7","Casseles","25"
,"Gravadores","24","CHR$ ,uso
do","26-27","CLEAR","10-27","C
LOAD,Dragon","14","CLS,explica
cao de","27","CODE, Spectrum",
"8"
```

Como você pode observar, o programa, inicialmente, muda a cor da tela e das bordas, e limpa a tela. Escolha as cores de sua preferência, mas procure evitar ■ habitual fundo negro com letras brancas. Qualquer outra coisa terá efeito melhor, simplesmente por ser diferente.

Não se esqueça, porém, de que ■ facilidade de leitura é indispensável. Assim, para ■ letras, opte por uma cor que contraste bem com ■ fundo. Fora isso, use seu bom gosto: algumas combinações são, sem dúvida, bem mais agradáveis que outras.

[illegible]

As linhas 20 e 30 colocam na tela ■ palavras "EDITORA NOVA CULTURAL APRESENTA". Observe que o nome da empresa fica na primeira linha e ■ palavra "APRESENTA", duas linhas abaixo. Esse detalhe garante ao texto muito maior clareza.

O programa utiliza o comando **PRINT AT** para posicionar ■ palavras no centro da tela. Tente descobrir quais deveriam ser os números dessa instrução por meio dos cálculos explicados anteriormente. Veja se eles conferem com os do programa!

Ao contrário do programa anterior, este posiciona ■ palavras no lugar adequado — detalhe que, como você pode notar, determina uma grande diferença de qualidade entre as telas.

Pode-se também utilizar ■ instrução **PRINT TAB** para posicionar ■ mesma palavra, mudando a linha 30 para:

30 PRINT 'TAB 10; INVERSE 1; "APRESENTA"

O apóstrofo junto ■ TAB 10 faz com que o Spectrum deixe uma linha em branco antes de imprimir "APRESENTA" ■ tela. Pode-se também obter o mesmo resultado simplesmente colocando uma instrução **PRINT** vazia (com um número de linha apropriado).

O programa faz ■ pausa após essa mensagem para dar maior ênfase ao próximo item que deverá ser impresso, o nome INPUT.

As linhas 50, 60 e 70 imprimem INPUT com uma borda que pisca ■ vermelho e amarelo. Essa borda ■ feita com caracteres gráficos da ROM — como ■ vê no programa — com o comando **FLASH** usando vermelho ■ amarelo.

Essas linhas utilizam uma série de comandos **PRINT AT** ■ mostram o quanto ele é flexível. Não há necessidade de ■ repetir o PRINT enquanto um ponto e vírgula é colocado entre os itens da linha.

Os vários **AT** são separados por ponto ■ vírgula. Caso os separássemos apenas por vírgulas, estas colocariam meia

linha de espaço na tela, o que poderia apagar alguma outra coisa que ali se encontrasse. Assim, ■ não ser que você queira apagar o que estiver nesta posição da tela, use de preferência ■ ponto ■ vírgula.

Como a borda piscante não fica no centro da linha, o cálculo de suas coordenadas não pode ser feito da maneira explicada anteriormente. Calcule as posições tomando as coordenadas da palavra central da linha ■ adicione ■ subtraia o número necessário para obter os resultados para a borda.

Tomemos, como exemplo, a primeira linha da borda. Já sabemos que as coordenadas da palavra INPUT são 10, 10. Sabemos também que esta linha fica uma acima de INPUT. Então, subtraindo 1 do primeiro número, obtemos a primeira coordenada: 9.

Como queremos que a borda comece um espaço antes da palavra, novamente subtraímos 1, agora da segunda coordenada de INPUT, obtendo 9.

Calcule os números das outras partes das bordas e compare seus resultados às coordenadas que empregamos.

O programa completa a tela com uma mensagem de direitos autorais (para lembrar aos usuários que é ilegal copiar programas) ■ informa que se deve pressionar qualquer tecla para continuar. Ao fazê-lo, não aparecerão instruções nem o início de um jogo, ■ um índice de INPUT.

Quando se pressiona uma tecla, o Spectrum limpa a tela ■ imprime a mensagem "INPUT — INDICE" em seu topo. A mensagem é posicionada no canto esquerdo da linha. Se quiser centralizá-la, introduza nove espaços entre as aspas e a primeira letra da mensagem. Em seguida, o computador deixa uma linha em branco, resultado do **PRINT** sem nenhuma mensagem da linha 130.

O laço **FOR...NEXT** que se segue lê duas variáveis alfanuméricas da linha **DATA** de número 180. Pode parecer estranho que números de páginas estejam armazenados em variáveis desse tipo. Observe, no entanto, que às vezes encontramos mais de um número. Sua armazenagem numa variável numérica seria interpretada como uma subtração e, quando imprimíssemos o valor da variável, iríamos obter o valor – 6, o que causaria muita confusão!

Além disso, alguns dados apresentam informações separadas por vírgula. Como você sabe, esta é a pontuação que separa diferentes itens numa linha **DATA**. Assim, ■ necessário colocar esses dados entre aspas para que o computador não os interprete como coisas distintas — com as aspas, ele lê a informação como um bloco, sem separá-la.

A linha 160 controla a impressão das informações da linha **DATA**, determinando sua posição na tela.

O **PRINT TAB** alinha os dados ■ números de páginas para que todos comecem na mesma coluna, dando à lista uma aparência organizada. Ele é muito útil na impressão de índices, determinando a coluna em que o dado começará a ser impresso. Nosso programa coloca a indicação na coluna 3 e a página na coluna 25.

Observe que usamos apenas um **PRINT** para imprimir os dois dados na linha, sendo que os **TAB** são separados por ponto e vírgula. Se você mudar o **TAB 25** para **TAB (25 + 32)** ou **TAB 57**, não notará nenhuma diferença. Quando ■ emprega um número maior que 32, ele é dividido por 32 e o resto é desprezado. O cursor não muda de linha até que ■ utilize ■ retrocesso — ou seja, até que um TAB tenha um valor menor do que ■ posição atual do cursor. O micro pula, então, para a linha seguinte e nela coloca a informação.

Experimente mudar os valores do TAB para ter ■ idéia melhor do que eles significam em termos de posição na tela. Lembre-se de que esta tem 32 caracteres por linha.

O **PRINT AT** e o **PRINT TAB**, embora muito parecidos, são usados em circunstâncias diferentes. Sempre que você quiser imprimir uma lista de palavras ou números na tela, utilize **PRINT TAB**. **PRINT AT** é mais poderoso para a impressão de mensagens isoladas em determinado ponto da tela.

T

Digite este programa e veja como o **PRINT@** pode ser usado para produzir uma página-título adequada:

```
10 CLS 3:B$=CHR$(128)
20 PRINT @71,B$;"nova";B$;"cult
ural";B$;
30 PRINT @138,"APRESENTA";
40 PRINT @358,B$;"copyright";B$
;B$;B$;"1986";B$;
50 PRINT @232,CHR$(190);STRING$
(11,CHR$(188));CHR$(189);
60 PRINT @264,CHR$(234);" I N ■
 U ■ ";CHR$(229);
70 PRINT @296,CHR$(155);STRING$
(11,CHR$(147));CHR$(151);
80 PRINT @448," QUALQUER TECLA
PARA CONTINUAR  ";
90 J=1-J:SCREEN 0,J
100 FOR K=1 TO 200:NEXT
110 IF INKEY$<>"" THEN 130
120 GOTO 50
130 CLS ■
140 PRINT @7,"INPUT - INDICE";
150 FOR X=3 TO 12
160 READ D$,E$
170 PRINT @32*X+1,D$;:PRINT @32
*X+25,LEFT$(" "+E$+"      ",7);
180 NEXT X
190 DATA ANIMACAO,26-32,"BASIC,
PROGRAMACAO",2-7,"BREAK,DRAGON"
,7,"CASSETE,FITAS",25,"CASSETE,
GRAVADORES",24,"CHR$,USO DE",26
-27,CLEAR,"10,27","CLOAD,TRS",1
4,"CLS,EXPLICACAO",27,"CODE,SPE
CTRUM",8
200 GOTO 200
```

O programa começa mudando ■ cor da tela para azul, na linha 10. O resto da linha iguala **B$** a um quadrado escuro. As linhas 20 ■ 40 imprimem as palavras em caracteres reversos, usando B$ como um espaço. Lembre-se de que os caracteres reversos do TRS-Color aparecem em minúsculo nas listagens.

A linha 30 é similar, com uma diferença: como a palavra APRESENTA é menos importante, aparece em caracteres maiúsculos normais.

No centro da tela está ■ título INPUT rodeado por um bloco gráfico colorido. Os caracteres e o título são impressos pelas linhas 50 a 70.

O manual indica os caracteres gráficos que estão disponíveis. Eles são constituídos por áreas verdes e áreas escuras. Adicionando-se um múltiplo de 16 ao código do caractere, o verde pode ser substituído por amarelo, azul, vermelho etc.

As áreas da tela coloridas de verde são aquelas em que a cor de fundo aparece. É fácil mudar a cor da tela do verde para o laranja e produzir um efeito pisca-pisca. Para a mudança, a linha 90 usa o comando **SCREEN** de maneira similar à utilizada para mudar a cor de gráficos de alta resolução. **SCREEN 0,1** muda a cor da tela para laranja e **SCREEN 0, 0** traz de volta o verde. O programa troca a cor da tela a cada execução do laço.

Para que se possa apreciar melhor a mudança de cor, a linha 100 estabelece uma pequena pausa. A linha 120 vai fechar o laço.

Se alguma tecla é pressionada enquanto a página está sendo mostrada, a linha 110 faz com que o programa passe para a parte seguinte.

A linha 130 muda a tela para vermelho. A linha 170 e o laço **FOR...NEXT** das linhas 150 e 180 encarregam-se de imprimir os dados lidos da linha 190. A cada execução do laço, uma nova linha é usada para a referência e o número de linha. O **LEFT$** faz com que o número da linha apareça sempre num painel do mesmo tamanho, o que assegura um visual bastante claro.

O toque final cabe ao laço da linha 200 que, embora não pareça, é bem importante. Sem essa linha, o programa colocaria uma mensagem de prontidão na tela, desfigurando nosso trabalho.

Vejamos uma tela mais cuidada para apresentar o nosso jogo:

```
10 COLOR 15,12,12:KEYOFF
20 SCREEN1:CLS:Z$=CHR$(1)
30 PRINTTAB(8);"NOVA CULTURAL"
40 PRINT:PRINTTAB(6);"A P R E S
 E N T A"
50 LOCATE 11,9:PRINTZ$+CHR$(88)
;:FORX=1TO5:PRINT Z$+CHR$(87);:
NEXT:PRINTZ$+CHR$(89)
60 LOCATE 11,10:PRINTZ$+CHR$(86
);"INPUT";Z$+CHR$(86)
70 LOCATE 11,11:PRINTZ$+CHR$(90
);:FORX=1TO5:PRINT Z$+CHR$(87);
:NEXT:PRINTZ$+CHR$(91)
80 LOCATE 5,19:PRINT"Copyright
(C) 1986"
90 LOCATE 2,22:PRINT"Tecle a ba
rra de espaços"
100 IF INKEY$=""THEN100
110 CLS:LOCATE 7:PRINTCHR$(207)
;"INPUT - INDICE";CHR$(208)
120 FOR J=1TO7: LOCATE 0,2*J+6
130 READ A$,B$
140 PRINTA$;TAB(23);B$
150 NEXT
160 GOTO 160
170 DATA ANIMAÇAO,26-32,PROGRAM
AÇAO BASIC,2-7,"CTRL-STOP, MSX"
,8,SISTEMA OPERACIONAL,25,CONTR
OLADORES DE DISCO,24,"CHR$, uso
 do",26-27,"POKE, como usar o",
27-28
```

O programa começa mudando a cor da tela para verde, com caracteres brancos, e desativando as indicações das teclas de função. Em seguida, seleciona a tela de texto de 32 colunas e define a variável **Z$** como o caractere de código 1. Adiante, veremos a utilidade dessa variável.

As linhas 30 e 40 colocam na tela, de forma organizada, a mensagem "NOVA CULTURAL APRESENTA". O **PRINT TAB**, com um valor adequado, encarrega-se de centralizar as palavras, enquanto o **PRINT** vazio que inicia a linha 40 produz o espaço de uma linha entre as palavras.

As linhas 50 a 70 imprimem o título INPUT dentro de uma moldura formada por caracteres gráficos. Esses caracteres são acessados pela instrução **PRINT** na forma: **PRINT CHR$(1) + CHR$** (código do caractere + 64).

Observe o uso da instrução **LOCATE**, já explicada anteriormente, para posicionar na tela caracteres gráficos e palavras.

A linha 100 pode parecer um tanto estranha, mas é muito importante no programa. Ela faz com que o computador varra repetidamente o teclado, até que pressionemos alguma tecla, passando para a parte seguinte.

A linha 110, por sua vez, limpa a tela e posiciona uma mensagem no topo desta. As referências e números de páginas são lidos da linha **DATA** e impressos pelas instruções do laço das linhas 120 a 150.

A linha 160, finalmente, tem como função evitar que o cursor e um OK provocados pelo término da execução do programa atrapalhem o visual da tela.

```
10  HOME
20  INVERSE : PRINT : HTAB 12:
PRINT " NOVA CULTURAL "
30  PRINT : HTAB 10: PRINT " A
P R E S E N T A "
40  VTAB 12: PRINT  SPC( 16): N
ORMAL : PRINT " INPUT ";: INVER
SE : PRINT  SPC( 17)
45  NORMAL : VTAB 20: HTAB 15:
PRINT "VERSAO 1.1"
50  PRINT : HTAB 11: PRINT "COP
YRIGHT (C) 1986"
60  FLASH : VTAB 24: HTAB 7: PR
INT " TECLE A BARRA DE ESPACOS"
;: NORMAL
70  GET A$
80  HOME : INVERSE : PRINT  SPC
( 13);"INPUT - INDICE"; SPC( 13
): NORMAL
90  FOR J = 1 TO 7: VTAB J * 2
+ 8
100  READ I$,P$: PRINT  TAB( 7)
;I$; TAB( 30);P$: NEXT
110  GOTO 110
120  DATA   ANIMACAO,26-32,PRO
GRAMACAO BASIC,2-7,"CTRL-C, App
le",7,SISTEMA OPERACIONAL,25,CO
NTROLADORES DE DISCO,24,"CHR$,
uso do",26-27,"POKE, como usar
o",27-28
```

O programa produz uma tela bem mais elaborada que a anterior. Logo que se inicia, ele limpa a tela e seleciona o modo invertido de apresentação de caracteres. A instrução **PRINT** vazia nas linhas 20 e 30 faz com que o computador pule uma linha antes de começar a imprimir as mensagens.

Os comandos **HTAB** e **VTAB**, como já explicamos, posicionam o cursor para a impressão.

Na linha 40, a instrução **PRINT SPC** (..), em modo invertido, provoca o aparecimento de uma linha escura antes do título INPUT, dando-lhe destaque.

As linhas 45 a 50 imprimem no modo normal mensagens de *copyright* e da versão do programa.

A linha 60 utiliza o comando **FLASH**. Os usuários do TK-2000 devem substituí-lo por **INVERSE**. A mensagem aparecerá piscando no Apple, juntamente com o cursor (é quase impossível escondê-lo, mas, dessa forma, ele se confunde com a mensagem).

A linha 70 faz o computador esperar até que alguma tecla seja pressionada para continuar a execução.

Em seguida, o programa limpa a tela novamente e coloca a mensagem "INPUT - INDICE" no modo invertido. O laço das linhas 90 e 100 lê e imprime na tela as referências e números de páginas respectivos. Observe que isso é feito por meio dos comandos **READ** e **PRINT TAB**. A mesma instrução **PRINT** imprime a referência e o número da página, estando os **TAB** separados por ponto e vírgula. Esse processo facilita muito a montagem de listas e tabelas.

A linha 110 impede que o programa termine. Assim, evita-se o aparecimento do cursor na tela, o que afetaria a distribuição das mensagens.

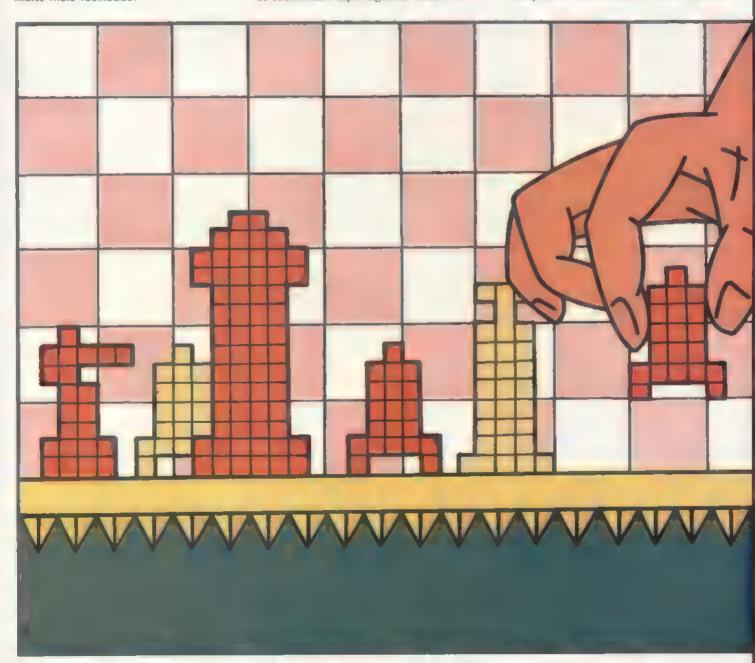
GERAÇÃO DE BLOCOS GRÁFICOS (2)

- NOVAS FUNÇÕES
- INVERSÃO DE CORES E FORMAS
- IMAGENS AO ESPELHO
- COMO RODAR OS CARACTERES
- O USO DA IMPRESSORA

Aqui está a [illegible] que faltava a seu programa gerador [illegible] blocos gráficos. Com as novas funções incorporadas, você poderá criar caracteres [illegible] muito mais facilidade.

Este artigo completa o programa gerador de caracteres gráficos. As novas linhas acrescentam-lhe recursos que tornam ainda mais fácil a criação de blocos definidos pelo usuário.

Além da parte final do programa, você encontrará aqui algumas dicas de como usar os caracteres criados em [illegible] programas BASIC.

Depois [illegible] acrescentar as [illegible] linhas

[illegible] programa dado [illegible] artigo anterior, você terá mais seis funções à sua disposição.

A primeira evidencia-se já no início. Ela mostra na tela os valores dos oito bytes correspondentes às linhas do bloco, bem como uma versão em tamanho real na tela. Isso é feito por um programa em linguagem de máquina, que atualiza tanto os valores como [illegible] bloco em tamanho real sempre que modificamos um ponto.

O programa completo também permite que você limpe [illegible] quadriculado por meio da tecla C, evitando o trabalho de apagar ponto por ponto.

Existem ainda três outras teclas de controle [illegible] que se pode recorrer para modificar o padrão do quadriculado. Se você apertar o M, o bloco será substituído por sua imagem ao espelho. Assim, sempre que quiser obter uma figura composta por dois blocos simétricos — uma espaçonave, por exemplo —, basta desenhar [illegible] deles, guardá-lo no banco de memória e criar a imagem simétrica usando a tecla M. Também lançamos mão desta função quando estamos desenhando duas figuras iguais, cada qual apontando para uma direção.

De modo semelhante, é possível rodar [illegible] bloco [illegible] graus, pressionando a tecla R. Esta função será muito útil quando se precisar criar apenas um bloco — de um torpedo, digamos — [illegible] depois virá-lo em todas as direções.

Com [illegible] última das três funções você obterá [illegible] inverso do caractere, por meio da tecla I. Se usá-la duas vezes seguidas, terá [illegible] inverso do inverso, ou seja, o caractere original.

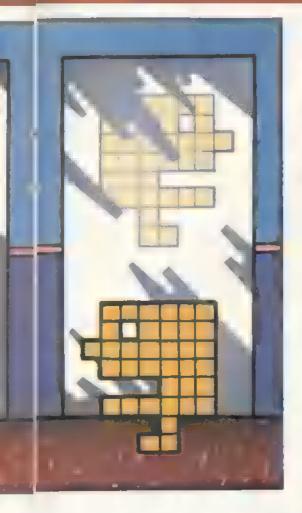
O programa passará [illegible] dispor, também, de [illegible] rotina de impressão. Pode-[illegible] empregá-la para imprimir os valores dos bytes que serão colocados em uma linha **DATA**, por exemplo, ou para fazer [illegible] cópia da tela. Esta sairá melhor ou pior de acordo com o tipo de [illegible] impressora.

Para usar a impressora, pressione [illegible] tecla Z, seguida de D ou S, conforme queira valores de bytes ou uma cópia da tela, respectivamente. Caso outra tecla seja utilizada, o programa voltará à edição.

Se você não tiver uma impressora, não copie [illegible] linhas de 2570 a 2590. Acrescente, porém:

```
2570 GOTO 2000
```

Aqui está o resto do programa:

```
   5 CLEAR 31999
  12 LET T=0: FOR N=32000 TO
  32227: READ A: POKE N,A: LET
  T=T+A: NEXT N: IF T<>21691
  THEN  PRINT  FLASH 1;"ERRO NA
  S LINHAS 'DATA'": STOP
2020 PRINT AT 10,21;CHR$ 139;CH
R$ 131;CHR$ 135:AT 11,21;CHR$ 1
38;AT 11,23;CHR$ 133;AT 12,21;C
HR$ 142;CHR$ 140;CHR$ 141
2030 RAND USR 32000
2530 IF INKEY$="I" THEN  [illegible] U
SR 32092
2540 IF INKEY$="C" THEN  POKE 3
2106,0: [illegible] USR 32092: POKE 32
106,12
2550 IF INKEY$="M" THEN  RAND U
[illegible] 32145
2560 IF INKEY$="R" THEN  [illegible] U
SR 32183
2570 IF INKEY$<>"Z" THEN  GOTO
2000
2575 INPUT "C(OPIA DA TELA) OU
B(YTES)? "; LINE Z$
```

```
2580 IF Z$<>"C" AND Z$<>"B" THE
█  GOTO 2000
2590 IF Z$="C" THEN  COPY : GOT
O 2000
2600 LET CH=65: FOR N=USR "A" T
O ███ "U"+7 STEP 8
2610 LET TA=0: LPRINT CHR$ CH:
FOR M=N TO N+7: LPRINT TAB TA;P
EEK M;: LET TA=TA+4: NEXT M
2620 LPRINT : LET CH=CH+1: NEXT
 █
9100 DATA 62,2,205,1,22,62,22,2
15,62,8,215,175,215,33,11,72,22
1,33,118,72
9110 DATA 6,8,197,6,8,14,128,17
5,50,91,125,126,254,1,40,7,58,9
1,125,129
9120 DATA 50,91,125,203,57,35,1
6,239,58,91,125,221,119,0,229,2
21,229,62,23,215
9130 DATA 62,5,215,33,90,125,20
5,40,26,62,13,215,221,225,225,1
7,24,0,25,221
9140 DATA 229,209,20,213,221,22
5,193,16,189,201,0,0,33,11,72,6
,8,197,6,8
9150 DATA 197,126,254,1,229,40,
12,6,7,62,1,119,36,16,252,54,25
5,24,13,6
9160 DATA 4,54,85,36,54,171,36,
16,248,37,54,255,225,193,35,16,
219,17,24,0
9170 DATA 25,193,16,209,201,33,
11,72,6,8,197,6,4,17,7,0,197,22
9,126,25
9180 DATA 78,119,225,113,35,27,
27,193,16,242,17,28,0,25,193,16
,229,205,92,125
9190 DATA 195,92,125,221,33,11,
74,33,235,72,6,8,197,6,8,197,22
1,126,0,119
9200 DATA 221,35,6,32,43,16,253
,193,16,241,17,24,0,221,25,17,1
,1,25,193
9210 DATA 16,226,205,92,125,195
,92,125
```

Os valores nas linhas **DATA** são programas em linguagem de máquina colocados no alto da memória com **POKE**. Ali existem três rotinas diferentes para rodar, espelhar ■ colocar o caractere em tamanho real na tela, junto com o valor dos bytes.

Tome cuidado na digitação desses valores. Qualquer erro pode ser fatal. Por isso mesmo, o programa verifica as linhas **DATA**, provocando uma interrupção caso detecte algum erro.

USO DOS BLOCOS EM OUTROS ███

O programa do artigo anterior já incluía funções de gravação ■ leitura em fita. Por meio delas, podemos dispor de uma versão permanente dos blocos criados. O uso correto deste programa permite a criação de muitos bancos cheios de novos caracteres.

O Spectrum é capaz de utilizar 21 UDG de cada vez. Se precisar de um número maior, você terá duas opções: criar vários bancos de memória, ■ redefinir ■ conjunto de blocos a todo instante. Explicaremos mais tarde como isso é feito, mas, agora, adiantaremos alguns detalhes.

Desde que saiba como modificar o "apontador de UDG", você pode utilizar quantos caracteres quiser. O apontador ■ uma variável interna do Spectrum que diz ao computador onde encontrar os bytes correspondentes aos blocos definidos pelo usuário.

Mesmo que tenha gravado os bytes criados pelo programa a partir de um determinado endereço, você poderá trazer este conjunto de caracteres de volta da fita para qualquer outro endereço. Assim, se quiser usar três diferentes bancos de caracteres ao mesmo tempo, bastará carregá-los em posições diferentes da memória.

Cada banco criado pelo programa tem 168 bytes de comprimento. Por isso, ■ necessário carregar cada conjunto de caracteres em posições que tenham uma distância de pelo menos 168 bytes entre si; caso contrário, um banco poderá apagar parte de outro. Para carregar um conjunto de caracteres gravado pelo programa, digite:

```
LOAD "" CODE (endereço inicial)
```

Independentemente do endereço a partir do qual o banco tenha sido gravado, pode-se colocá-lo em qualquer parte da RAM.

Para usar os bancos que carregou, modifique o valor do apontador de UDG. Em artigo posterior, trataremos do assunto em detalhes.

É importante adiantar, ainda, que ao usarmos mais de um banco de caracteres em um programa deveremos proteger, por meio de um comando **CLEAR**, a área de memória ocupada. Isso também será explicado mais tarde.

Feitas as modificações, o programa terá várias novas funções. Você notará a primeira delas quando for guardar um bloco recém-criado no banco: o progra■ imprimirá, ao lado do quadriculado, ■ valor do byte correspondente ao padrão de cada linha do bloco, acompanhado do valor do byte de cor e do código da cor de frente e de fundo daquela linha.

O programa completo permite ainda que você limpe o quadriculado por meio da tecla A, evitando o trabalho de apagar ponto por ponto.

Outra alternativa interessante que o programa oferece ■ a de se obter, com a tecla E, ■ imagem ao espelho do bloco que está no quadriculado. Essa função facilita bastante ■ produção de desenhos simétricos.

Muito útil também é ■ possibilidade de rodar o bloco 90 graus, o que nos possibilita criar um torpedo, por exemplo, e apontá-lo em todas ■ direções, usando a tecla V.

Com a tecla I, pode-se inverter todo o padrão do bloco. Pressionando-a, os pontos ███ se apagam e os outros se acendem. A tecla Y tem o mesmo efeito, mas troca a cor de frente com a cor de fundo, não atuando, assim, ■ estado dos pontos. Em outras palavras, o I modifica apenas os bytes do padrão do bloco, enquanto o Y modifica os bytes de cor — e esta ■ uma diferença muito importante, embora não se reflita no efeito obtido. A última função utiliza uma impressora para reproduzir o padrão do bloco que está no quadriculado, bem como os valores dos bytes de padrão e cor. Para imprimir, pressiona-se a tecla L.

```
40 GOSUB 6000
135 IF K$="V" THEN GOSUB 4000:G
OTO 200
145 IF K$="A" THEN GOSUB 4500:G
OTO 200
155 IF K$="Y" THEN GOSUB 5000:G
OTO 200
165 IF K$="L" THEN GOSUB 5500:G
OTO 200
180 IF K$="I" THEN GOSUB 3000:G
OTO 200
190 IF K$="E" THEN GOSUB 3500:G
OTO 200
1550 GOSUB 9000:FOR I=0 TO 7:P(
I)=0:FOR J=0 TO 7
1595 II=64+I*8:PRESET(12,II):PR
INT#1,P(I)
1605 PRESET(56,II):PRINT#1,C(I)
:PRESET(176,II):PRINT#1,(C(I)-(
C(I)AND15))/16:PRESET(220,II):P
RINT#1,C(I)AND15
1610 NEXT I
1615 II=II+10:PRESET(12,II):PRI
NT#1,"BYTE"
1620 PRESET(56,II):PRINT#1,"COR
"
1630 PRESET(164,II):PRINT#1,"FR
ENTE"
1640 PRESET(216,II):PRINT#1,"FU
NDO"
1650 PRESET(8,160):PRINT#1,"Qua
lquer tecla para continuar"
1660 X$=INKEY$:IF X$="" THEN 16
60
1670 LINE(0,64)-(88,160),15,BF
1680 LINE(164,64)-(255,160),15,
BF
1690 GOSUB 9100:RETURN
3000 FOR I=0 TO 7:FOR J=0 TO 7
3010 B(I,J)=B(I,J)XOR1
3020 GOTO 1080
3500 FOR I=0 TO 7:FOR J=0 TO 7
3510 T(I,J)=B(I,7-J)
3520 NEXT J,I:FOR I=0 TO 7:FOR
J=0 TO 7
3530 B(I,J)=T(I,J):GOTO 1080
4000 FOR I=0 TO 7:FOR J=0 TO 7
4010 T(I,J)=B(J,7-I)
4020 GOTO 3520
4500 FOR I=0 TO 7:FOR J=0 TO 7
4510 B(I,J)=0
4520 GOTO 1080
5000 FOR I=0 TO 7
5010 Y1=C(I)AND15
5020 Y2=(C(I)-Y1)/16
5030 C(I)=Y1*16+Y2
5040 NEXT I:FOR I=0 TO 7:FOR J=
0 TO 7
5050 GOTO 1080
5500 LPRINT TAB(10);"BYTE";
5510 LPRINT TAB(15);" COR";
5520 LPRINT TAB(20);"FRENTE";
5530 LPRINT TAB(28);"FUNDO"
5550 FOR I=0 TO 7:P(I)=0:FOR J=
0 TO 7
5560 P(I)=P(I)+B(I,J)*2^(7-J)
5570 IF B(I,J)=1 THEN LPRINT "X
"; ELSE LPRINT ".";
5580 NEXT J
5590 LPRINT TAB(10);P(I);
5600 LPRINT TAB(16);C(I);
5610 LPRINT TAB(22);(C(I)-(C(I)
AND15))/16;
5620 LPRINT TAB(28);C(I)AND15
5630 NEXT I
5640 LPRINT:LPRINT:LPRINT
5650 RETURN
6000 FOR I=0 TO 7
6010 C(I)=C*16+F:NEXT
6020 RETURN
```

Não há função automática que permita o aproveitamento do banco de blocos gravado em fita. Para isso, precisamos da ajuda de algumas linhas escritas ■ BASIC.

A elaboração de um programa desse tipo exige que se conheça bem ■ organização da memória de vídeo do MSX. Em artigo futuro trataremos deste assunto em detalhes.

ALGUMAS INFORMAÇÕES

Para os usuários mais avançados, adiantamos que um banco gravado pelo programa pode ser lido à qualquer momento. Inicialmente, deve-se proteger o alto da memória com:

```
CLEAR 200, &HD000
```

Em seguida, carrega-se o banco digitando:

```
BLOAD "CAS:"
```

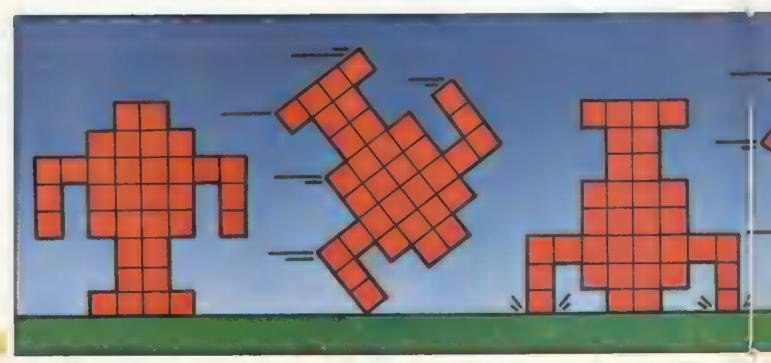
Com isso, ■ banco volta à mesma posição de memória onde foi criado. Os oito bytes correspondentes ■ padrão do bloco guardado na posição ■ podem, então, ser encontrados ■ partir do seguinte endereço:

```
&HD100 + X*8
```

E os oito bytes de cor do bloco guardado na posição **X** do banco se encontram ■ partir de:

```
&HE100 + X*8
```

O passo seguinte consiste em transferir, com ■ ajuda de **VPOKE**, os bytes de padrão para ■ tabela de padrões — **BASE(12)** — ■ os bytes de cor para ■ ta-

bela de cores — **BASE(11)**. Maiores esclarecimentos serão dados num próximo artigo.

O programa conta agora com várias outras funções de edição. Uma delas é a de limpeza do quadriculado, disponível pela tecla C. Já não será preciso, portanto, apagar ponto por ponto.

A tecla U tem função semelhante, mas apaga apenas o oitavo bit, que controla o grupo de cores.

Duas outras novas funções contribuem para diminuir um pouco as dificuldades. Pressionando a tecla J, todo o padrão se desloca para a esquerda; a letra K tem efeito contrário. Utilizando essas teclas, pontos que estavam em colunas ímpares passam a ocupar colunas pares e vice-versa. Não havendo dois pontos adjacentes, o procedimento inverterá a cor dos pontos.

Para criar figuras simétricas, temos a tecla E, que produz a imagem em espelho do bloco que está no quadriculado. Ao usá-la, tenha sempre em mente as regras de definição de cores. Para rodar o padrão do quadriculado 90 graus, pressione a tecla V. Essas funções devem ser combinadas com outras, uma vez que modificam o oitavo bit, que não participa do padrão, mas da determinação de cores.

Dispomos ainda de duas funções de inversão: apagar o que está aceso e vice-versa. A tecla I inverte todo o bloco, enquanto a tecla O se restringe ao oitavo bit. O programa também exige os valores dos bytes, para quem quiser usá-los em linhas **DATA**. A tecla D mostra uma linha com os dados de um bloco, no rodapé da tela. Tendo uma impressora, os usuários do Apple podem obter uma cópia usando W.

```
40  DIM T(8,8)
135  IF K$ = "V" THEN  GOSUB 40
00: GOTO 50
145  IF K$ = "U" THEN LL = 7: G
OSUB 2500: GOTO 50
155  IF K$ = "J" THEN  GOSUB 45
00: GOTO 50
160  IF K$ = "I" THEN LL = 0: G
OSUB 3000: GOTO 50
165  IF K$ = "K" THEN  GOSUB 50
00: GOTO 50
170  IF K$ = "C" THEN LL = 0: G
OSUB 2500: GOTO 50
175  IF K$ = "D" THEN  GOSUB 60
00: GOTO 50
180  IF K$ = "O" THEN LL = 7: G
OSUB 3000: GOTO 50
185  IF K$ = "W" THEN  PR# 1: G
OTO 6000: PR# 0: GOTO 50
190  IF K$ = "E" THEN  GOSUB 35
00: GOTO 50
2500  FOR I = 0 TO 7: FOR J = L
L TO 7
2510 B(I,J) = 0: GOTO 1020
3000  FOR I = 0 TO 7
3005  FOR J = LL TO 7
3010  IF B(I,J) = 0 THEN B(I,J)
 = 1: GOTO 1020
3020 B(I,J) = 0: GOTO 1020
3500  FOR I = 0 TO 7: FOR J = 0
 TO 6
```

```
3510 T(I,J) = B(I,6 - J): NEXT
J,I
3520  FOR I = 0 TO 7: FOR J = 0
 TO 6
3530 B(I,J) = T(I,J): GOTO 1020
4000  FOR I = 0 TO 7: FOR J = 0
 TO 7
4010 T(I,J) = B(J,7 - I): NEXT
J,I
4020  FOR I = 0 TO 7: FOR J = 0
 TO 7
4030 B(I,J) = T(I,J): GOTO 1020
4500  FOR I = 0 TO 7: FOR J = 0
 TO 6
4510 B(I,J) = B(I,J + 1): GOTO
1020
5000  FOR I = 0 TO 7: FOR J = 0
 TO 6
5010 T(I,7 - J) = B(I,6 - J): N
EXT J,I
5020  FOR I = 0 TO 7: FOR J = 0
 TO 6
5030 B(I,J) = T(I,J): GOTO 1020
6000  HOME : VTAB 22: INPUT "QU
AL O NUMERO DO BLOCO ?";N: IF N
 > 320 THEN 1000
6020  HOME
6030  VTAB 21: PRINT "DATA ";
6040  FOR I = 0 TO 7: PRINT  PE
EK (E + N * 8 + I);: IF J <  >
7 THEN  PRINT ",";
6050  NEXT : VTAB 23: GET X$
6060  HOME : RETURN
```

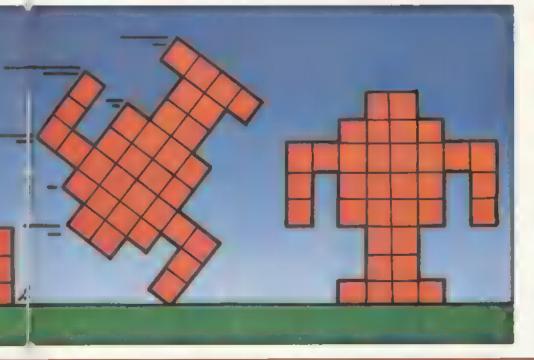
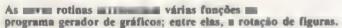
USO DOS BLOCOS EM OUTROS PROGRAMAS

Além dos problemas na exibição de cores, o Apple tem uma organização de memória de vídeo um tanto complicada. Deixaremos, assim, para um artigo futuro as explicações sobre o uso dos blocos em outros programas.

No artigo anterior vimos a primeira metade de um programa cujo objetivo é facilitar a criação de blocos gráficos. Apresentamos aqui a outra metade, que acrescenta ao programa novos recursos de edição

Inicialmente, carregue o programa antigo e adicione as linhas:

```
30 T=0:FOR K=0 TO 14:READ N:T=T
+N:POKE K+31100,N:NEXT:READC:IF
 T<>C THEN END
40 T=0: FOR K=0 TO 84:READ N:T=
T+N:POKE 31150+K,N:NEXT:READ C:
IF T<>C THEN END
70 DATA 141,72,142,123,12,236,1
29,237,193,140,123,84,38,247,57
,1974
80 DATA 141,22,142,123,84,51,67
,198,3,166,130,167,194,90,38,24
9,51,70,140,123,12,38,240,57,18
9,179,237,52
90 DATA 6,31,3,142,123,14,134,3
,183,121,68,230,192,134,8,74,88
```

As [illegible] rotinas [illegible] várias funções [illegible] programa gerador de gráficos; entre elas, a rotação de figuras.

Com o programa completo, você poderá também inverter seus gráficos, utilizando a cor do fundo da tela.

```
,73,125,121,23,39,4,88,73,128,4
,68
100 DATA 102,132,125,121,23,39,
3,68,102,132,77,38,230,48,31,12
2,121,68,38,219,48,6,140,123,86
,38,207,53,192,8285
170 PRINT"JOYSTICK OU TECLADO (
J OU T)?";
180 A$=INKEY$:IF A$<>"J" AND A$
<>"T" THEN 180
190 IF A$="J" THEN JY=1
350 IF JY=1 THEN GOSUB 1000 ELS
E GOSUB 1500
1000 PUT(X1,Y1)-(X1+5*T-1,Y1+4)
,C1,NOT
1010 IF (PEEK(65280)AND 1)=0 GO
SUB 2000
1020 IF JOYSTK(1)=0 THEN Y=Y-1
1030 IF JOYSTK(1)=63 THEN Y=Y+1
1040 IF JOYSTK(0)=0 THEN X=X-T
1050 IF JOYSTK(0)=63 THEN X=X+T
1060 RETURN
2100 GET(216,70)-(239,93),A
2110 GOSUB 3000:GOTO 2070
2200 A$=INKEY$:IF A$<>"S" AND A
$<>"P" THEN 2200
2210 CLS:DN=0:IF A$="P" THEN DN
=-2
2220 FOR K=0 TO 14:FOR R=0 TO 2
2230 PRINT #DN,PEEK(VARPTR(A(0)
)+K*3+R);:NEXT:PRINT#DN:NEXT
2240 IF DN<0 THEN 2260
2250 A$=INKEY$:IF A$="" THEN 22
50
2260 FOR K=15 TO 23: FOR P=0 TO
 2
2270 PRINT #DN,PEEK(VARPTR(A(0)
)+K*3+R);:NEXT:PRINT #DN:NEXT
2280 A$=INKEY$:IF A$="" AND DN=
0 THEN 2280
2290 SCREEN 1,ST:RETURN
2800 POKE 30999,T-1:N=USR2(VARP
TR(A(0)))
2810 GOSUB 3000:GOTO 2070
2900 POKE 30999,T-1:N=USR1(VARP
TR(A(0)))
2910 GOSUB 3000:GOTO 2070
```

Complete a linha 10 com:

```
:DEFUSR1=31100:DEFUSR2=31150
```

Se executarmos o programa agora, poderemos utilizar as novas funções. Mas é conveniente, antes de qualquer coisa, verificar se há algum erro em linhas **DATA**.

Três funções foram somadas ao repertório do programa: espelhar, inverter e rodar o bloco gráfico.

Para obter a imagem do bloco ao espelho, basta pressionar a tecla M. O caractere é invertido da esquerda para a direita. Essa função é muito útil quando utilizamos dois blocos justapostos para produzir uma figura maior e simétrica. Basta criar um dos caracteres, guardá-lo no banco e, em seguida, obter a imagem no espelho.

A função de rotação é similar. Apertando R, rodamos o bloco [illegible] graus. Podemos, assim, mover a figura para cima e para baixo, o que nos permite usar o gerador para criar uma versão do bloco "de cabeça para baixo".

A inversão da cor de todos os pontos constitui outra nova opção oferecida pelo programa. Em **PMODE4**, o efeito é fácil de prever: o preto se torna verde (ou cinza) e vice-versa. Em **PMODE3**, azul e amarelo são invertidos (o que é azul fica amarelo e vice-versa). O mesmo ocorre com vermelho e verde, cinza e laranja, ciano e magenta.

As inversões de cor provocam alterações surpreendentes no aspecto de certos blocos. Uma vez que nos acostumemos [illegible] as regras de mudança de cor, a função será muito útil.

É possível, também, executar [illegible] mudança de cor durante a edição, pressionando-se a tecla V. A alteração, no caso, resulta de uma inversão no tipo de tela — **SCREEN** — que estiver sendo utilizada.

Outro novo recurso do programa, a impressão dos valores dos bytes dos caracteres guardados no banco, mostra-se especialmente útil quando se pretende colocar tais valores em linhas **DATA**. A tecla P ativa a função; os valores podem ser exibidos na tela ou, então, copiados por uma impressora. Após teclarmos P, o programa espera que pressionemos P ou S. P utiliza a impressora e S, a tela.

USO DOS [illegible] EM OUTROS PROGRAMAS

Para utilizar os blocos gravados em fita dentro de um outro programa, carregue o conteúdo da fita e, em seguida, guarde [illegible] padrões [illegible] matrizes com **CLOADM** e **GET**.

Infelizmente, você precisará repetir o procedimento sempre que quiser usar o programa com os blocos gráficos anteriormente gravados.

Para evitar o trabalho de ler os blocos da fita todas as vezes, só há uma alternativa: transferir os números correspondentes [illegible] blocos para linhas **DATA** e incorporá-las ao programa em questão.

Assim, embora não pareça, pode ser mais conveniente recorrer à função de impressão para digitar linhas **DATA**. Os números ali contidos deverão ser colocados pelo programa, usando **POKE**, na tela, e transferidos para matrizes por comandos **GET**. Depois disso, os blocos estarão disponíveis por meio de [illegible]dos **PUT**.

O BASIC NA MEMÓRIA

- COMO É ARMAZENADO UM PROGRAMA EM BASIC?
- O USO DO PEEK
- O QUE SIGNIFICAM OS CÓDIGOS

Como o computador programa BASIC na memória? Satisfaça sua curiosidade: com um programa bem simples, você poderá listar os códigos secretos usados por micro.

Já tivemos a oportunidade de examinar como organiza o espaço total de memória dos microcomputadores (veja artigo da página 174). Em geral este espaço divide-se internamente em uma série de áreas, cada qual com uma função específica.

Uma dessas áreas da memória RAM é reservada para os programas em BASIC do usuário. Eles começam sempre em um mesmo endereço absoluto, cujo valor varia conforme linha do micro. Assim, quando computador recebe um comando **RUN**, **LIST** ou **LLIST**, por exemplo, já "sabe" onde o programa se inicia, podendo começar a listá-lo ou interpretá-lo. Em muitos computadores, esse endereço de início está contido em um par de apontadores da RAM, em um lugar prefixado. Alterando-se seus valores com alguns **POKE**, pode-se mudar o local da memória onde o programa em BASIC está armazenado.

A listagem dos códigos internos que seu microcomputador utiliza para armazenar esses programas constitui um ótimo exercício para quem está começando a aprender programação em linguagem de máquina.

O PROGRAMA

Neste artigo, apresentaremos um programa bem simples, baseado no comando **PEEK**, que nos permite examinar os códigos numéricos decimais contidos nas locações das memórias ROM ou RAM, e seu equivalente em ASCII. Você poderá, assim, identificar as partes dos programas que contêm códigos diferentes do ASCII.

Os computadores de algumas linhas já dispõem de programas-monitores residentes, que servem perfeitamente para este propósito (é o caso do TRS-80, do Apple II do TK-2000). Por isso, programas seguintes destinam-se apenas aos micros das linhas ZX-81, Spectrum, TRS-Color e MSX, que não dispõem desse recurso.

T MSX

```
10 FOR I=7681 TO 7851 STEP 5
20 PRINT I;TAB(6);":";
30 FOR J=I TO I+4:PRINT USING
"### ";PEEK(J);:NEXT J
60 PRINT TAB(25);
70 FOR J=I TO I+4
80 IF PEEK(J)>31 AND PEEK(J)<
129 PRINT CHR$(PEEK(J)); ELSE
PRINT " ";
120 NEXT J:PRINT:NEXT I
```

O programa acima funciona no TRS-Color. Para o MSX, modifique linha 10 para:

```
10 FOR I=-32765 TO -32595 STEP
   5
```

S

```
10 FOR i=23755 TO 23925 STEP
5
20 PRINT i;TAB 6;":";
30 FOR j=I TO i+4:PRINT PEEK
j;:NEXT j
60 PRINT TAB(25);
70 FOR j=i TO i+4
   IF PEEK j>31 AND PEEK j<96
THEN GOTO 110
90 PRINT CHR$ PEEK(j);:GOTO 120
110 PRINT " ";
120 NEXT j
130 PRINT
140 NEXT i
```

S

```
10 FOR I=16509 TO 16679 STEP
5
20 PRINT I;TAB 6;":";
30 FOR J=I TO I+4
40 PRINT PEEK J;
50 NEXT J
60 PRINT TAB 25;
70 FOR J=I TO I+4
80 IF PEEK J>31 AND PEEK J<96
THEN GOTO 110
90 PRINT CHR$ PEEK(J);
100 GOTO 120
110 PRINT " ";
120 NEXT J
130 PRINT
140 NEXT I
```

O laço **FOR...NEXT** que começa na linha 10 coloca o endereço absoluto da localização da memória a ser listada, a partir do ponto inicial de todo programa BASIC, em incrementos de 5. A linha 20 imprime o endereço absoluto da primeira memória de um grupo de 5, enquanto os laços das linhas 30 e 70 se encarregam, respectivamente, da impressão do conteúdo numérico dos caracteres ASCII correspondentes.

Observe que as palavras-chave do BASIC não são armazenadas caractere por caractere, mas sim como códigos numéricos de 1 byte — os chamados *tokens*, em inglês. Cada código corresponde uma das palavras reservadas do BASIC, existentes no manual do micro. Eles são traduzidos de volta para as palavras-chave quando o programa é listado com **LIST** ou **LLIST**. Armazenam-se em ASCII, por sua vez, constantes, nomes de variáveis, caracteres de pontuação, símbolos matemáticos e tudo o que estiver entre aspas.

Como exercício, procure identificar os códigos das palavras reservadas no programa listado.

O DIVERTIDO JOGO DA COBRA

Ajude uma cobra faminta ■ encontrar alimento. Enquanto vai abocanhando os números que o computador coloca na tela, o desnutrido animal se transforma, ■■ poucos, numa gigantesca serpente.

O jogo da cobra consiste num tipo clássico de videogame, de regras simples mas muito interessante ■ cheio de surpresas. Além disso, sua programação dispensa o uso de linguagem de máquina — ■ jogo desafia o tempo, sendo um dos mais facilmente programáveis em BASIC.

O JOGO

O objetivo do jogo ■ ajudar ■■■ cobra faminta ■ encontrar ■ alimento espalhado ■■ ■■■■ pela tela, na forma de números ou blocos. O valor desses números diminui ■ medida que o tempo passa; assim, quanto mais lento for o jogador, menor seu total de pontos. Se ele demorar demais, o número que estava na tela chega a zero e desaparece; outro número surge em posição diferente. Cada número engolido pela serpente aumenta seu comprimento proporcionalmente ao valor ingerido.

Devemos tomar cuidado para não deixar ■ cobra ultrapassar a moldura da tela ou passar sobre si mesma — o que fica cada vez mais difícil, conforme ela cresce. Se cruzarmos ■ margem ou o corpo da cobra, o jogo termina. A palavra "FIM" ou "FORA" é, então, exibida várias vezes na tela.

```
 10 BORDER 1: PAPER 7: INK 9:
 CLS
 20 LET hs=0
 30 DIM s(570,2)
100 LET s=0
110 LET s(1,1)=10: LET s(1,2)=
14
120 LET s(2,1)=11: LET s(2,2)=
14
130 LET s(3,1)=12: LET s(3,2)=
14
135 GOSUB 1500
140 LET t=1: LET h=3
145 LET yv=1: LET xv=0
150 FOR n=1 TO 3: PRINT  PAPER
4;AT s(n,1),s(n,2);"#": NEXT n
160 LET y=12: LET x=14
165 LET p=0
170 GOSUB 1000
190 IF ATTR (y,x)<>56 AND ATTR
(y,x)<128 THEN  GOTO 2000
200 LET h=h+1: IF h=501 THEN
LET h=1
210 PRINT  PAPER 4;AT y,x;"#"
220 LET s(h,1)=y: LET s(h,2)=x
230 PRINT AT s(t,1),s(t,2);
CHR$ 32
240 LET t=t+1: IF t=501 THEN
LET t=1
250 IF p=0 THEN  LET p=INT (
RND*9)+1: LET fy=INT (RND*19)+
2: LET fx=INT (RND*30)+1: IF
ATTR (fy,fx)<>56 THEN  LET p=0
: GOTO 250
260 PRINT  PAPER INT (p/2);
FLASH 1;AT fy,fx;p
270 IF RND<.98 THEN  GOTO 290
280 LET p=p-1: IF p=0 ■■■■
PRINT AT fy,fx;CHR$ 32
290 IF y<>fy OR x<>fx
THEN  GOTO 170
300 LET s=s+p: PRINT
PAPER 4;AT y,x;"#": PR
INT  PAPER 6;AT 0,6;s
310 FOR n=1 TO ■
320 GOSUB 1000
325 IF ATTR (y,x)<>56
THEN  GOTO 2000
330 LET h=h+1: IF h=501
THEN  LET h=1
340 LET s(h,1)=y: LET
s(h,2)=x
350 PRINT  PAPER 4;AT s(h,1),s
(h,2);"#"
355 FOR m=1 TO 10: NEXT m
360 NEXT n
500 GOTO 165
```

- UMA VERSÃO BASIC DO JOGO DA COBRA
- ESPALHE OS NÚMEROS E FAÇA A COBRA ABOCANHÁ-LOS
- COMO A FIGURA CRESCE

```
1000 LET a$=INKEY$
1010 IF a$="q" THEN  LET yv=-1:
 LET xv=0
1020 IF a$="a" THEN  LET yv=1:
LET xv=0
1030 IF a$="o" THEN  LET xv=-1:
 LET yv=0
1040 IF a$="p" THEN  LET xv=1:
LET yv=0
1050 LET y=y+yv: LET x=x+xv: RE
TURN
1500 FOR n=22560 TO 225
91: POKE n,16: POKE n+6
40,16: NEXT n
1510 FOR n=22592 TO 225
92+32*19 STEP 32: POKE
 n,16: POKE n+31,16: NE
XT n
1520 PRINT  PAPER 1;AT 0,0;"SCO
RE "; PAPER 6;s; PAPER 1;TAB 14
;"RECORDE  "; PAPER 6;hs; PAPER
 1;TAB 31;" "
1590 RETURN
2000 PRINT AT 0,0;: FOR n=1 TO
88: PRINT  FLASH 1; PAPER 2; IN
K 6;"FORA"; PAPER 6; INK 2;"FOR
A";: SOUND .005,60-n: NEXT n
2010 IF s>hs THEN  LET hs=s
2020 CLS : PRINT AT 8,10;"SCORE
: ";s;AT 11,8;"RECORDE: ";hs
2030 PRINT  INVERSE 1;AT 16,2;"
Qualquer tecla para recomecar"
2040 PAUSE 0: CLS : GOTO 100
```

A linha 10 cuida das cores da moldura, do fundo e dos caracteres. Também limpa a tela. As linhas 20 e 100 zeram o recorde e o score.

A linha 30 DIMensiona a matriz S, usada para armazenar as coordenadas da cobra. Inicialmente, esta ficará nas coordenadas (10,14), (11,14), (12,14), colocada nos primeiros elementos da matriz pelas linhas 110 e 130.

A linha 135 chama uma sub-rotina que prepara a tela — linha 1500. Para o desenho da moldura, os valores são colocados na tabela de atributos com o auxílio de **POKE**. A linha 1500 traça duas linhas na tela: uma em cima e outra embaixo; a 1510 desenha as linhas laterais. A linha 1520 escreve o recorde e o score.

A linha 140 determina onde estão a cabeça e o rabo da cobra — h e t, respectivamente — dentro da matriz s. Essas variáveis indicam ao computador os elementos da matriz em que as coordenadas da cabeça e do final do corpo da

cobra podem ser encontradas. Elas são chamadas de "apontadores" (*pointers*). Conforme ■ cobra se move, o computador reajusta os dois apontadores e coloca as novas coordenadas no elemento adequado da matriz.

MOVIMENTO E DIREÇÃO

As variáveis **xv** ■ **yv** indicam a direção em que está indo a cobra. Elas podem assumir três valores diferentes: 0, quando a cobra não está indo naquela direção; 1, quando ■ cobra se dirige para a direita ou para baixo; e − 1, quando a cobra está seguindo para a esquerda ou para cima. Pode-se ver, desse modo, que a linha 145 faz com que ■ serpente siga para cima no começo do jogo.

A linha 150 desenha a cobra, inicialmente, com apenas três caracteres. A posição atual da cabeça é dada por **x** e **y**, variáveis também usadas para detectar colisões. A linha 160 posiciona a cabeça em (14,12). A linha 165 faz com que ■ valor do número ■ ser mostrado — **p** — seja zero.

A linha 170 chama a sub-rotina que movimenta o cursor — linha 1000. Q e A movimentam a serpente na vertical, enquanto O e P o fazem na horizontal. Pressionando essas teclas, alteramos os valores de **xv** ■ **yv**; a linha 1050 modifica ■ posição da cabeça adicionando **xv** e **yv** a **x** e **y**.

Quando termina a sub-rotina, o programa retorna para a linha 190, que usa o comando **ATTR** para verificar se a cabeça está sendo colocada em uma posição que não é de cor branca — a cor do fundo — ou não ■ cintilante — como são os números. Se essas duas condições não forem satisfeitas, a cobra deve ter cruzado a moldura ou seu próprio corpo. Neste caso, o programa vai para a linha 2000, início da sub-rotina que informa o final do jogo.

Se a nova posição da cabeça for válida, o programa prossegue na linha 200, que incrementa o apontador da cabeça. Se o apontador indica um elemento da matriz maior do que sua dimensão, o apontador passa a valer 1. A linha 210 desenha a cabeça em sua nova posição e a linha 220 coloca as coordenadas desta posição na matriz s, no elemento indicado pelo apontador da cabeça — **h**. O rabo da cobra é mantido no tamanho correto por meio da impressão de espaços em branco que apagam os caracteres das posições já ocupadas. Isso é feito pela linha 230, que obtém ■ coordenadas corretas na matriz **s**. A linha 240, por sua vez, aumenta o valor do apontador da cauda — **t** — verificando se ele ultrapassou a dimensão da matriz.

Se não houver um número ■ tela para a cobra comer, a linha 250 escolhe um — **p** —, com posição e valor aleatórios. Se as coordenadas escolhidas não correspondem a ■ área em branco da tela — **ATTR < > 56** —, o valor e ■ posição de **p** são escolhidos novamente. A linha 260 escreve o número na tela cintilando, isto é, com atributo **FLASH**.

À medida que o tempo passa, o valor do número que está na tela vai diminuindo. A linha 270, que compara um número aleatório com 0.98, introduz um certo elemento de ■ na velocidade com que ocorre a diminuição. Na maioria dos casos, a linha que diminui o valor do número — 280 — é pulada. Essa linha verifica se ■ valor de **p** ainda não chegou a zero, logo após subtrair-lhe 1 — em caso afirmativo, um espaço em branco é colocado ■ lugar do número ■ tela. Se ■ cabeça da cobra não está ■ mesma posição que ■ número, completa-se o laço e ■ programa volta à linha 170.

Se a serpente engolir o número, o programa segue na linha 300, que soma o número engolido ao score. Este é colocado na tela, assim ■ ■ cabeça.

O laço **FOR...NEXT** das linhas 310 ■ 360 adiciona ■ corpo da cobra uma quantidade de caracteres igual ao valor ingerido. A cada volta do laço, a linha 320 chama a rotina de movimentação do cursor; a linha 325 verifica se a cabeça ocupa uma posição legal; ■ linha 330 incrementa o valor do apontador da cabeça; a linha 340 coloca a nova posição na matriz s, e a linha 350 imprime ■ cabeça. Nenhum caractere é apagado, como aconteceria normalmente, servindo todos para acrescentar segmentos ao corpo da cobra. Como estes também nunca são apagados, ■ cobra teria movimentos bem mais rápidos, se não houvesse ■ atraso provocado pela linha 355.

A porção final do programa — que começa na linha 2000 — é ■ sub-rotina que cuida do encerramento do jogo. A linha 2000 escreve vários "FORA" na tela, ao mesmo tempo que emite uma série de sons. A linha 2010 atualiza o valor do recorde, se este foi batido, ■ ■ linha 2020 informa o placar. As linhas 2030 e 2040 permitem que ■ jogador inicie uma nova partida.

```
5 SCREEN 1:KEY OFF
10 M=512:DIMB(M)
15 RR=RND(-TIME)
20 L=3:GOSUB 600
30 GOTO 500
100 K=STICK(0):IF K=0 THEN K=L
110 NP=B(H)+32*(K=1)-32*(K=5)-(
K=3)+(K=7)
120 TE=VPEEK(NP)
130 IF TE=255 OR TE=HC OR TE=BC
 OR NP<BASE(5)+32 THEN VPOKE NP
,HC:E=1
140 L=K:FOR I=1 TO 50:NEXT:RETU
RN
200 R=INT(RND(1)*9)+1:RX=INT(RN
D(1)*13)+2:RY=INT(RND(1)*29)+2:
RP=RX*32+RY
210 IF VPEEK(BASE(5)+RP)<>32 TH
EN 200
220 VPOKE BASE(5)+RP,48+R:P=1:R
ETURN
250 R=R-1:IF R=0 THEN VPOKE BAS
E(5)+RP,32:P=0:RETURN ELSE VPOK
E BASE(5)+RP,48+R:RETURN
300 SC=SC+R:LOCATE 22,0:PRINT S
C;:P=0
310 SL=SL+1:VPOKE NP,BC
320 H=H-1:IF H=0 THEN H=M
330 B(H)=NP
340 FOR J=1 TO R
350 PLAY"L64CEF"
360 GOSUB 100:IF E=1 THEN 800
370 H=H-1:IF H=0 THEN H=M
380 B(H)=NP:VPOKE NP,BC
390 NEXT
400 VPOKE NP,HC:RETURN
500 IF P=0 THEN GOSUB 200 ELSE
IF INT(RND(1)*150)+1<SL THEN GO
SUB 250
510 GOSUB 100:IF E=1 THEN 800
520 IF TE=48+R THEN VPOKE B(H),
BC:GOSUB 300
530 VPOKE B(H),BC:VPOKE B(T),32
540 H=H-1:IF H=0 THEN H=M
550 T=T-1:IF T=0 THEN T=M
560 B(H)=NP:VPOKE NP,HC
570 GOTO 500
600 BG=INT(RND(1)*13)+1:FG=INT(
RND(1)*13+1):P=0:SC=0:SL=1:E=0
610 IF BG=FG THEN 600
620 COLOR 15,BG,BG:CLS
630 VPOKE BASE(6)+4,240+BG
640 VPOKE BASE(6)+31,FG*16+FG
650 FOR J=0 TO 31:PLAY"T255L640
4AG":VPOKE BASE(5)+J,255:VPOKE
BASE(5)+J+736,255:NEXT:FOR J=32
 TO 736 STEP 32:PLAY"O2DA":VPOK
E BASE(5)+J,255:VPOKE BASE(5)+3
1+J,255:NEXT
660 LOCATE 3,0:PRINT"RECORDE=";
HS;:PRINT TAB(15);"SCORE = 0 ";
670 HC=2:BC=215:T=3:H=1
680 VPOKE BASE(6),FG*16+BG
690 VPOKE BASE(6)+26,FG*16+BG
700 B(1)= BASE(5)+239:VPOKE B(1
),HC
710 B(2)=B(1)+32:VPOKE B(2),BC
720 B(3)=B(2)+32:VPOKE B(3),BC
730 RETURN
800 COLOR 15,INT(RND(1)*14)+1:C
LS:PLAY"O1ACDEFGACDEFG":FOR K=1
 TO 408:PRINT "FIM ";:NEXT:PLAY
"O4ABCEDFGO3ABCDEFGO2ABCDEFG"
810 IF HS<SC THEN HS=SC
820 COLOR 1,15,15:CLS:LOCATE 9,
8:PRINT"SCORE =";SC:LOCATE 8,12
:PRINT"RECORDE =";HS
830 A$=INKEY$:LOCATE 3,20:PRINT
"<RETURN> para continuar"
```

```
840 A$=INKEY$:IF A$<>CHR$(13) T
HEN 840 ELSE 20
```

Ao contrário de outros jogos vistos em INPUT, para os quais utilizamos a tela gráfica, optamos aqui pela tela de textos de 32 colunas, já que ■ cobra é feita de caracteres.

A linha 5 seleciona ■ tela ■ desliga o menu das teclas de funções da parte inferior da tela. A linha 10 dimensiona a matriz ■ de maneira que esta acomode no máximo 512 caracteres — o maior tamanho que ■ cobra poderá ter. Note que ■ elementos dessa matriz não correspondem às posições de caracteres na tela. Cada um deles contém as coordenadas de uma parte da cobra. A linha 15 "dá a partida" ■ gerador de números randômicos; a 20 manda o programa para ■ linha 600, que prepara as condições iniciais do jogo.

Na linha 600, **BG** é a cor de fundo da tela e **FG**, a cor da moldura, bem como da cobra. **P=0** significa que nenhum número está sendo exibido; **SC=0**, que o score ■ zero; **SL=0** quer dizer que o nível de dificuldade é zero e **E=0** informa ■ computador que a cobra permanece dentro dos limites da tela, não tendo cruzado nem a moldura, nem seu próprio corpo. Todas essas variáveis são utilizadas ao longo do programa para que certas sub-rotinas sejam chamadas no momento adequado.

A linha 610 verifica se ■ cor de fundo é igual à da moldura — em caso afirmativo, duas novas cores são escolhidas. A linha 620 seleciona ■ cores da tela — caracteres brancos, fundo ■ bordas de acordo com **BG**. Além disso, limpa a tela.

Na linha 630, um comando **VPOKE** assegura que o caractere de espaço em branco — código 32 — tenha cor de frente branca e cor de fundo igual ■ **BG**.

Na linha 640, a mesma técnica ■ empregada para fazer com que o caractere 255, usado para desenhar ■ moldura, tenha cor de frente e de fundo igual a **FG**. A linha 650 desenha ■ moldura, enquanto emite alguns sons. A linha 660 mostra o placar.

Na linha 670, **HC** é ■ código do caractere da cabeça da cobra e **BC**, o código do caractere usado no corpo da mesma. As variáveis **H** e **T** indicam os elementos de **B()** que contêm as coordenadas da cabeça e da ponta da cauda da serpente. São chamadas de "apontadores" (*pointers*).

Ainda usando o comando **VPOKE** para modificar os valores da tabela de cores da tela de 32 colunas, as linhas 680 e 690 fazem com que os caracteres da cabeça e do corpo da cobra tenham cor de frente igual ■ **FG** e cor de fundo igual a **BG**. As linhas 700 a 720 colocam a cabeça ■ dois segmentos do corpo na tela, de modo que a cobra tenha apenas estas partes ■ cada início de partida.

ROTINA PRINCIPAL

Da sub-rotina o programa retorna ■ linha 30, que o envia para a rotina principal, ■ linha 500. Essa linha verifica, inicialmente, se na tela há um número para ■ cobra comer. **P** é o indicador de número na tela; se seu valor for zero, a sub-rotina da linha 200 é chamada. Se houver um número na tela, um valor qualquer entre 1 e 150 é comparado com o nível de dificuldade **SL**. Se o número gerado for menor que **SL**, o programa vai à sub-rotina 250.

A sub-rotina que começa na linha 200 coloca um número qualquer na tela, em uma posição escolhida ao acaso. A linha 200 seleciona o número — R — entre 1 ■ 9, bem como suas coordenadas **RX** ■ **RY**. Esses dois valores são usados para calcular **RP**, posição do número na tabela de nomes da tela 1. Antes que o número seja impresso na tela, a linha 210 verifica se na sua posição há algum outro caractere além do de espaço — código 32. Dessa maneira, evita que o número seja colocado sobre a moldura ou sobre ■ corpo da serpente. A linha 220 escreve o número na tela, usando **VPOKE** para colocar o código correspondente ao número na tabela de nomes — **BASE (5)** — da tela 1. Consultando uma tabela ASCII, você verá que os caracteres de 0 a 9 têm códigos de ■ a 57.

A sub-rotina que começa na linha 250 faz o valor do número na tela ir diminuindo até que o jogador leve ■ cobra até ele. Se o número chegar a zero antes da cobra comê-lo, um espaço em branco será colocado em seu lugar. Enquanto isso não acontecer, valores sucessivamente mais baixos substituem os anteriores, ■ mesma posição. A sub-rotina retorna para a linha 510.

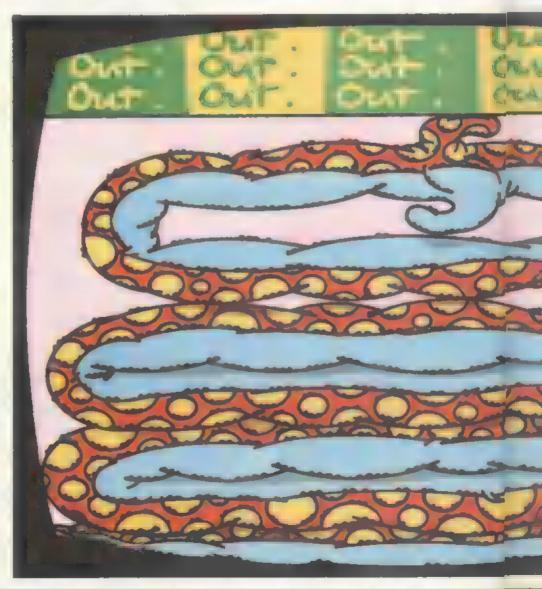
Essa linha chama outra sub-rotina, na linha 100, que recebe os comandos de movimentação do cursor. Podemos usar tanto ■ teclas com setas como um joystick, desde que ■ comando **STICK** da linha 100 seja modificado (veja o artigo da página 348). O **IF...THEN** verifica se a cobra cruzou ■ moldura ou seu próprio corpo. Em caso afirmativo, o programa vai para a linha 800, que cuida do fim da partida.

A linha 520 testa se ■ número foi engolido pela cobra, somando 48 a **R** e comparando o resultado com **TE**. Somamos 48 ■ ■ para obter ■ código do caractere correspondente ■ número que estava na tela. **TE**, obtido com **VPEEK**, corresponde ao código do caractere que estava na posição que a cabeça da cobra ocupa agora. Caso o número tenha sido realmente engolido, um caractere do corpo é colocado em sua posição. O programa vai, então, para a sub-rotina

da linha 300, que calcula o novo score, aumenta o tamanho da cobra proporcionalmente ao número engolido — laço entre as linhas 340 [illegible] 390 — [illegible] coloca outro número [illegible] tela.

As linhas 540 e 550 modificam [illegible] apontadores da cabeça e da ponta da cauda, ajustando [illegible] valores, caso tenham se reduzido a zero. A linha 560 coloca o caractere da cabeça na tela.

A rotina que cuida do encerramento [illegible] partida começa na linha 800, que limpa a tela, usando [illegible] cor ao acaso, e produz um som enquanto a palavra "FIM" é impressa várias vezes. No fim da operação, outro som é produzido. A linha 810 atualiza o recorde, antes que as linhas 820 a 840 ofereçam ao jogador a oportunidade de jogar novamente.

```
5  [illegible] : GR
10 M = 500: DIM B(M)
20  GOSUB 600
30  GOTO 500
100  GET K$: IF K$ < > "P" [illegible]
 K$ < > "L" AND K$ < > "X" AN
D K$ < > "Z" THEN K$ = L$
110 NP = B(H) - 40 * (K$ = "P")
 + 40 * (K$ = "L") - (K$ = "Z")
 + (K$ = "X")
120 TE =  SCRN( [illegible] -  INT (NP /
 40) * 40, INT (NP / 40))
130  IF TE = 9 [illegible] TE = 11 OR TE
 = 8 OR NP < 40 THEN  COLOR= 11
: PLOT NP -  INT (NP / 40) * 40
, INT (NP / 40):E = 1
140 L$ = K$: RETURN
200 R =  INT ( RND (1) * 9) + 1
:RX =  INT ( RND (1) * 37) + 2:
RY =  INT ( [illegible] (1) * 37) + [illegible]
210  IF  SCRN( RX,RY) [illegible] > [illegible] TH
EN 200
220  VTAB 22: HTAB 31: PRINT "B
ONUS ";R:P = [illegible]
230  COLOR= 6: PLOT RX,RY: RETU
RN
250 [illegible] = R - 1: IF [illegible] = [illegible] THEN  C
OLOR= 0: PLOT RX,RY:P = 0: RETU
RN
260  HTAB 31: VTAB 22: PRINT "B
ONUS ";R: RETURN
300 SC = SC + R: HTAB 7: VTAB 2
2: PRINT SC:P = [illegible]
310 SL = SL * 1: COLOR= 8: PLOT
 NP -  INT (NP / 40) * 40, INT
 (NP / 40)
320 H = [illegible] - 1: IF H = [illegible] THEN H
= M
330 B(H) = NP
340  FOR J = 1 TO R
350 XX =  PEEK ( - 16336)
360  GOSUB 100: IF E = 1 [illegible] 8
00
370 H = [illegible] - 1: IF H = [illegible] THEN H
= M
380 B(H) = NP: COLOR= 8: PLOT N
P -  INT (NP / 40) * 40, INT (N
P / 40)
390  NEXT
400  COLOR= 11: PLOT [illegible] -  INT
(NP / 40) * 40, [illegible] (NP / 40)
410  RETURN
500  IF P = [illegible] [illegible] [illegible] 200:
GOTO 510
505  IF  INT ( [illegible] (1) * 150) +
 1 < [illegible] THEN [illegible] 250
510 [illegible] 100: IF E = 1 THEN [illegible]
[illegible]
520  IF [illegible] = 6 [illegible] COLOR= 0:
PLOT B(H) - [illegible] (B(H) / 40) *
40, INT (B(H) / 40): GOSUB 300
530  COLOR= 8: PLOT B(H) -  INT
 (B(H) / 40) * 40, INT (B(H) /
40): COLOR= 0: PLOT B(T) -  INT
 (B(T) / 40) * 40, INT (B(T) /
40)
540 H = [illegible] - 1: IF H = 0 [illegible] [illegible]
= M
550 T = T - 1: IF T = 0 THEN T
= M
560 B(H) = NP: COLOR= 11: PLOT
NP -  INT (NP / 40) * 40, INT (
[illegible] / 40)
570  GOTO 500
600 L$ = "X":P = 0:SC = 0:SL =
1:E = 0
650  COLOR= 9: HLIN 0,39 AT 0:
VLIN 0,39 AT 0: VLIN 0,39 AT 39
: HLIN 0,39 AT 39
660 [illegible] : VTAB 22: PRINT "SCO
RE ";SC: TAB( 15);"RECORDE ";HS
670 T = 3:H = 1: COLOR= 11
680 B(1) = 300: PLOT B(1) -  IN
T (B(1) / 40) * 40, INT (B(1) /
 40)
690  COLOR= 8
700 B(2) = 340: PLOT B(2) -  IN
T (B(2) / 40) * 40, INT (B(2) /
 40)
710 B(3) = 380: PLOT B(3) - [illegible]
T (B(3) / 40) * 40, INT (B(3) /
 40)
720 [illegible]
[illegible]  HGR : TEXT : [illegible] : FOR I
= 1 TO 500:XX =  PEEK ( - 16336
```

```
): PRINT "FIM  ";: NEXT
805  IF HS < SC THEN HS = SC
810  HOME : HTAB 15: VTAB 8: PR
INT "SCORE = ";SC: HTAB 14: VTA
B 12: PRINT "RECORDE = ";HS
820  HTAB 6: VTAB 20: PRINT "<R
ETURN> PARA JOGAR NOVAMENTE": G
ET A$
830  IF A$ =  CHR$ (13) THEN  H
OME : GR : GOTO 20
880  GOTO 820
```

Para o jogo no Apple ■ no TK-2000, escolhemos a tela de baixa resolução, já que ■ cobra é feita de blocos.

A linha 5 seleciona ■ tela e limpa ■ parte inferior. A 10 dimensiona ■ matriz B de modo a acomodar no máximo quinhentos caracteres — o maior tamanho que ■ cobra poderá ter. Note que os elementos da matriz B não correspondem às posições dos blocos na tela. Cada um deles contém as coordenadas de uma parte da cobra. A linha 20 manda ■ programa para ■ linha 600, que prepara as condições iniciais do jogo.

Na linha 600, **P**=0 significa que nenhum bloco está sendo mostrado; **SC**=0, que o score é zero; **SL**=0 quer dizer que o nível de dificuldade é zero e **E**=0 informa ao computador que a cobra permanece dentro dos limites da tela, não tendo cruzado nem a moldura, nem seu próprio corpo. Todas essas variáveis são utilizadas ao longo do programa para que certas sub-rotinas sejam chamadas no momento adequado.

A linha 650 desenha a moldura. A linha 660 mostra o placar.

Na linha 670, 11 é a cor da cabeça da cobra. **H** e **T** são variáveis usadas para indicar os elementos de **B ( )** que contêm as coordenadas da cabeça e da ponta da cauda da serpente. Essas duas variáveis são chamadas de "apontadores" (*pointers*).

As linhas 680 a 710 colocam a cabeça e dois segmentos do corpo na tela, de modo que ■ cobra tenha apenas estas partes a cada início de partida. A cor do corpo da cobra é 8.

■ NA TELA

Da sub-rotina o programa retorna à linha 30, que o envia para ■ rotina principal, na linha 500. Essa linha verifica, inicialmente, se há comida (bloco) para a cobra na tela. ■ ■ o indicador de bloco na tela; ■ seu valor for zero, a sub-rotina da linha 200 ■ chamada. Se houver um bloco na tela, um número qualquer entre 1 ■ 150 é comparado com o nível de dificuldade **SL**. Se o número gerado for menor que **SL**, ■ linha 505 manda o programa para ■ sub-rotina 250.

A sub-rotina que começa ■ linha ■ coloca um bloco na tela, em uma posição escolhida ao acaso. A linha 200 seleciona ■ número — R — entre 1 ■ 9, bem como duas coordenadas ■ e **RY**. A linha 210 verifica se ■ posição tem ■ cor 0, preta, indicando que não ■ nada no local. A linha 220 imprime na porção inferior da tela ■ valor do bloco que ali está (bônus). A linha 230 coloca um bloco de cor ■ em um ponto qualquer da tela.

A sub-rotina da linha 250 diminui gradativamente o valor do bônus, até que o jogador consiga fazer ■ cobra chegar à comida. Se o número for reduzido ■ zero antes que ■ cobra coma o bloco, um espaço em branco ■ colocado em seu lugar. Enquanto isso não ocorre, valores mais baixos são sucessivamente exibidos. A sub-rotina retorna para a linha 510.

Esta linha chama outra sub-rotina, na linha 100, que recebe os comandos de movimentação do cursor. Usamos ■ teclas P, L, Z e X como de costume. O **IF...THEN** verifica se a cobra cruzou a moldura ou seu próprio corpo; em caso afirmativo, o programa vai para a linha 800, que cuida do encerramento da partida.

A linha 520 testa se ■ comida foi engolida pela cobra, comparando a cor do bloco com **TE**. Usamos **SCRN** na linha 120 para obter **TE**, cor do bloco que estava ■ posição que a cabeça da cobra ocupa. Caso a comida tenha sido realmente engolida, um bloco do corpo ■ colocado em ■ posição. O programa vai, então, para ■ sub-rotina da linha 300, que calcula o novo score, aumenta o tamanho da cobra proporcionalmente ao bônus — laço entre as linhas 340 ■ 390 — e, por fim, coloca outro bloco na tela.

As linhas 540 e 550 mudam os apontadores da cabeça ■ da ponta da cauda, ajustando seus valores, caso tenham ■ reduzido a zero. A linha 560 coloca o bloco da cabeça da cobra na tela.

A rotina que cuida do final da partida começa na linha 800, que limpa a tela e provoca um ruído (no Apple), enquanto ■ palavra "FIM" ■ impressa várias vezes. A linha 805 atualiza ■ recorde, antes que as linhas 810 a 830 ofereçam ao jogador ■ oportunidade de jogar novamente.

T

```
10 M=512:DIM B(M)
20 GOSUB 600
30 GOTO 500
100 K$=INKEY$:IF K$="" THEN K$=
L$
110 K=ASC(K$):NP=B(H)-32*(K=10)
```

```
+32*(K=94)-(K=9)+(K=8)
120 TE=PEEK(NP)
130 IF TE=FG OR TE=HC OR TE=BC OR NP<1056 THEN POKE NP,HC:E=1
140 LS=KS:RETURN
200 R=RND(9):RX=RND(13)+1:RY=RND(29)+1:RP=RX*32+RY
210 IF PEEK(1024+RP)<>BG THEN 200
220 R$=RIGHT$(STR$(R),1):PRINT@RP,R$;:P=1:RETURN
250 R=R-1:IF R=0 THEN PRINT @RP,CHR$(BG);:P=0:RETURN ELSE PRINT @RP,RIGHT$(STR$(R),1);:RETURN
300 SC=SC+R:PRINT @26,SC;:P=0
310 SL=SL+1:POKE NP,BC
320 H=H-1:IF H=0 THEN H=M
330 B(H)=NP
340 FOR J=1 TO R
350 PLAY "L255CEF"
360 GOSUB 100:IF E=1 THEN 800
370 H=H-1:IF H=0 THEN H=M
380 B(H)=NP:POKE NP,BC
390 NEXT
400 POKE NP,HC:RETURN
500 IF P=0 THEN GOSUB 200 ELSE IF RND(159)<SL THEN GOSUB 250
510 GOSUB 100: IF E=1 THEN 800
520 IF TE=R+112 THEN POKE B(H),BC: GOSUB 300
530 POKE B(H),BC:POKE B(T),BG
540 H=H-1:IF H=0 THEN H=M
550 T=T-1:IF T=0 THEN T=M
560 B(H)=NP:POKE NP,HC
570 GOTO 500
600 BG=RND(7)+1:FG=RND(7)+1:L$=CHR$(94):P=0:SC=0:SL=1:E=0
610 IF BG=FG THEN 600
620 CLS BG:BG=BG-1:FG=FG-1
630 BG=143+16*BG
640 FG=143+16*FG
650 FOR J=1024 TO 1055:PLAY "T255L255O4AG":POKE J,FG:POKE J+480,FG:NEXT:FOR J=1056 TO 1472 STEP 32:PLAY"O2DA":POKE J,FG:POKE J+31,FG:NEXT
660 PRINT @3,"RECORDE=";HS;:PRINT @15,"SCORE= 0 ";
670 HC=ASC("*")+64:BC=ASC("#")+64:T=3:H=1
680 B(1)=1263:POKE B(1),HC
690 B(2)=1295:POKE B(2),BC
700 B(3)=1327:POKE B(3),BC
710 RETURN
800 CLS RND(8): PLAY"O1ACDEFGACDEFG":FOR K=1 TO 408:PRINT " FORA!! ";:NEXT:PLAY"O4ABCDEFGO3ABCDEFGO2ABCDEFG"
810 IF HS<SC THEN HS=SC
820 CLS:PRINT @73,"SCORE    =";SC:PRINT @230,"   RECORDE =";HS
830 A$=INKEY$:PRINT @450,"PRESSIONE QUALQUER TECLA PARA   RECOMECAR"
840 A$=INKEY$: IF A$="" THEN 840 ELSE 20
```

Usamos a tela de textos, e não a gráfica, para o jogo no TRS-Color, já que a cobra é feita de caracteres.

A linha 10 dimensiona a matriz B de modo que esta acomode no máximo 512 caracteres — o maior tamanho que a cobra poderá ter. Note que os elementos da matriz B não correspondem às posições de caracteres na tela. Cada elemento contém as coordenadas de uma parte da cobra. A linha 20 manda o programa à linha 600, que prepara as condições iniciais do jogo.

Na linha 600, **BG** é a cor de fundo da tela e **FG**, a cor da moldura. **L$ = CHR$(94)** faz com que a cobra se movimente para cima, até que uma tecla seja pressionada. **P = 0** significa que nenhum número está sendo mostrado; **SC = 0**, que o score é zero; **SL = 0** quer dizer que o nível de dificuldade é zero, e **E = 0** informa ao computador que a cobra permanece dentro dos limites da tela, não tendo cruzado nem a moldura, nem seu próprio corpo. Todas essas variáveis são utilizadas ao longo do programa para que certas sub-rotinas sejam chamadas no momento adequado.

A linha 610 verifica se a cor de fundo é igual à da moldura, escolhendo duas novas cores se o par inicial for idêntico. A linha 620 usa **CLS BG** para colorir a tela. Subtrai 1 de **BG** e **FG**, numa preparação para as contas das linhas 630 e 640. Embora pareçam complexas, essas contas simplesmente convertem **BG** e **FG** em valores que possam ser colocados na tela com **POKE**, a fim de produzir a cor desejada.

A linha 650 desenha a moldura, enquanto produz alguns sons. A linha 660 mostra o placar.

Na linha 670, **HC** é o código do caractere da cabeça da cobra e **BC**, o código do caractere usado no corpo da mesma. **H** e **T** são variáveis usadas para indicar os elementos de **B( )** que contêm as coordenadas da cabeça e da ponta da cauda da serpente. Essas duas variáveis são chamadas de "apontadores" (*pointers*).

As linhas 680 a 700 colocam na tela a cabeça e dois segmentos do corpo, de modo que a cobra tenha apenas estas partes a cada início de partida.

Da sub-rotina o programa retorna para a linha 30, que o envia para a rotina principal na linha 500. Essa linha verifica, inicialmente, se há um número na tela para a cobra comer. **P** é o indicador de número na tela. Se seu valor for igual a zero, a sub-rotina da linha 200 é chamada. Se houver um número na tela, um número qualquer entre 1 e 150 é comparado com o nível de dificuldade **SL**. Se o número gerado for menor que **SL**, o programa vai à sub-rotina 250.

A sub-rotina que começa na linha 200 coloca na tela um número qualquer, em uma posição escolhida ao acaso. A linha 200 seleciona o número — R — entre 1 e 9, bem como as coordenadas **RX** e **RY**. Esses dois valores são usados para calcular **RP**, posição do número na tabela de nomes da tela 1. Antes que o número seja impresso na tela, a linha 210 verifica se na sua posição há outra cor além da cor de fundo, para evitar uma superposição com a moldura ou com o corpo da serpente. À linha 220 cabe escrever o número na tela, e colocar 1 em **P**.

A sub-rotina que começa na linha 250 diminui gradativamente o valor do número na tela, até que o jogador consiga fazer a cobra comê-lo. Se o número for reduzido a zero, um espaço em branco é colocado em seu lugar. Enquanto isso não ocorre, valores sucessivamente mais baixos são colocados na mesma posição. A sub-rotina retorna para a linha 510.

Essa linha chama outra sub-rotina, na linha 100, que recebe os comandos de movimentação do cursor. O **IF...THEN** verifica se a cobra cruzou a moldura ou seu próprio corpo; em caso afirmativo, o programa vai para a linha 800, que cuida do fim da partida.

A COBRA

A linha 520 testa se a cobra engoliu o número, somando 112 a R e comparando o resultado com **TE**. Somamos 112 a R para obter o código do caractere correspondente ao número que estava na tela. **TE** é obtido com **PEEK**, e corresponde ao código do caractere que estava na posição que a cabeça da cobra ocupa agora. Caso o número tenha sido realmente engolido, um caractere do corpo é colocado em sua posição. O programa vai, então, para a sub-rotina da linha 300, que calcula o novo score, aumenta o tamanho da cobra proporcionalmente ao número engolido — laço entre as linhas 340 e 390 — e coloca outro número na tela.

As linhas 540 e 550 modificam os apontadores da cabeça e da ponta da cauda, ajustando seus valores, caso eles tenham se reduzido a zero. A linha 560 coloca o caractere da cabeça da cobra na tela.

A rotina que cuida do encerramento da partida começa na linha 800, que limpa a tela usando uma cor ao acaso e produz algum som enquanto a palavra "FIM" é impressa várias vezes. Ao final da operação, outro som é produzido. A linha 810 se incumbe de atualizar o recorde, antes que as linhas 820 a 840 ofereçam ao jogador a oportunidade de jogar novamente.

COMO ESCOLHER UMA IMPRESSORA

- ESCOLHA O TIPO CERTO DE IMPRESSORA
- PAPEL PARA IMPRESSÃO
- COMO CONECTAR A IMPRESSORA AO SEU COMPUTADOR

Se você já desejou algum dia uma listagem impressa de seus programas, ou uma cópia dos gráficos que aparecem na tela, chegou o momento de adquirir uma impressora.

Periférico dos mais úteis para um microcomputador, uma impressora contribui não só para a concretização de muitas aplicações novas, como também para um notável progresso na produtividade e na eficiência da programação. Entretanto, como ela representa um investimento considerável, e como existe no mercado uma grande variedade de tipos e preços, é necessário planejar com bastante cuidado a aquisição do modelo mais adequado.

Embora a impressora possa ser vista apenas como um periférico de saída alternativo para o vídeo, isto não quer dizer que ambos tenham as mesmas funções. O papel principal da impressora é assegurar um registro permanente de qualquer informação armazenada ou processada pelo computador, mesmo que esta nunca seja exibida na tela. Mas ela pode proporcionar também cópias da tela em papel (o que é conhecido como *hardcopy*, em jargão técnico).

Há muitos tipos de programas capazes de desenhar na tela gráficos científicos (chamados também de "arte computacional") que o usuário gostaria de copiar, arquivar, ou até colocar em uma moldura. O mesmo acontece com os desenhos técnicos produzidos no vídeo por software de projetos (CAD ou *Computer-Aided-Design*). Tudo isso pode ser reproduzido no papel por uma impressora gráfica adequada.

Certas aplicações, porém, exigem apenas uma impressora de textos: por exemplo, se você trabalha com programas de bancos de dados ou com progra-

Acionadas por solenóides, as agulhas instaladas na "cabeça" da impressora matricial pressionam a fita tintada contra o papel para formar os caracteres.

mas contábeis, que não necessitem de gráficos, pode encontrar uma impressora mais barata, e igualmente adequada.

Um computador doméstico pode ser usado de modo semelhante a uma máquina de escrever: seu teclado permite datilografar cartas, artigos, livros etc. Existem programas de processamento de textos que manipulam com rapidez e eficiência palavras, linhas e parágrafos. Nesse tipo de aplicação, a impressora um periférico essencial.

A impressora funciona ainda como elemento de apoio para a atividade de programação. Neste particular, os méritos do vídeo são muitos, mas as pessoas costumam ter mais familiaridade palavra impressa. Por outro lado, a maioria dos programadores gosta de ter cópia impressa do programa; esta não só dá uma idéia global do programa, como facilita o trabalho de detecção correção de erros. Além disso, é sempre bom contar uma listagem do programa em papel para manter um arquivo paralelo.

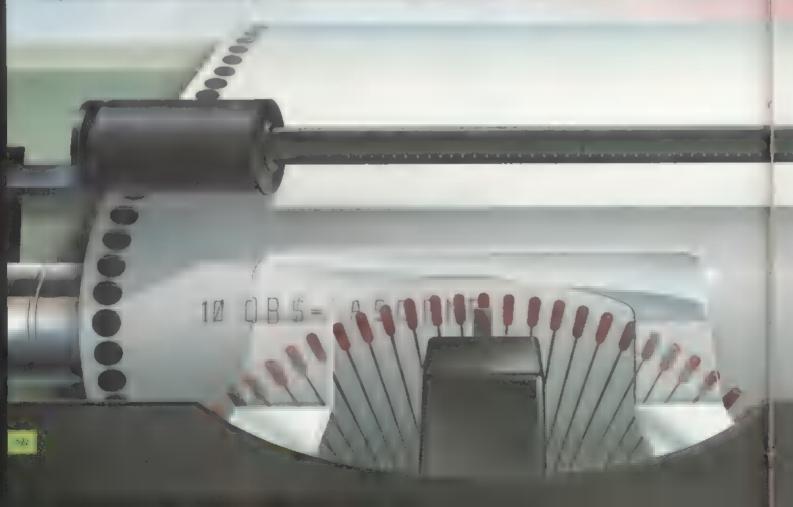
TIPOS DE IMPRESSORA

No Brasil, usuário de computadores pessoais conta com três espécies de impressora: a matricial de impacto, a do tipo "margarida" e a impressora térmica. Um quarto modelo é impressora-traçadora, muito utilizada na Europa no Japão; essa espécie, porém, é encontrada no Brasil apenas parte do equipamento do microcomputador de bolso PC-1500, da Sharp.

Como acontece com todas as impressoras modernas, a matricial conta carro de papel fixo. A cabeça de impres- movimenta no sentido transversal sobre o papel; ela contém uma colu- vertical de agulhas bem próximas outras e ligadas a indutores do tipo solenóide. Quando solenóide é ativado, a agulha propulsionada contra uma fita tintada, imprimindo um ponto papel. Os caracteres são formados por meio da combinação de agulhas, e da movimentação da cabeça de impressão.

As primeiras impressoras matriciais tinham matrizes de 5 x 7 (cinco colunas por sete fileiras), porém as mais modernas já utilizam agulhas, para produzir um padrão de 7 9 ou 9 x 9. Este é um aspecto importante, pois qualidade final dos caracteres impressos depende do número de pontos na matriz. Uma das características "negativas" das impressoras matriciais, quando operam com matrizes menores, é a impossibilidade de traçar parte da letra que fica abaixo da linha de impressão, como nas letras p q.

Essas impressoras são as mais rápidas de todas, existindo modelos com velocidades que variam de 80 a 400 cps (caracteres por segundo). A velocidade máxima, entanto, raramente é atingida. As impressoras mais modernas imprimem linhas de texto nas duas direções (impressão bidirecional).

Mesmo melhores modelos de impressoras matriciais, os caracteres impressos, embora claramente legíveis, são nitidamente formados por pontinhos. Entretanto, possível conseguir uma qualidade melhor de apresentação. O recurso mais utilizado para isso é impressão múltipla, que faz com que cabeça imprima duas vezes cada linha de texto: uma vez em cada direção. Antes da segunda passagem, rolo de papel deslocado em uma fração de milímetro, o que um pequeno desalinhamento entre as agulhas de impressão e os caracteres já impressos. Assim, os espaços em branco deixados na primeira passagem são preenchidos na segunda, e os caracteres ficam com um aspecto final mais completo mais escuro (qualidade carta).

Esse recurso, porém, reduz a velocidade normal de impressão à metade e tem efeito apenas em impressoras matriciais com capacidade gráfica. Nestas, o computador controla não só o disparo individual de cada agulha, como o grau de deslocamento horizontal da cabeça de impressão e vertical do rolo.

Existe ainda um tipo de cabeça com um conjunto duplo de agulhas. Esse dispositivo não diminui a velocidade de impressão, mas é menos usado por ser mais caro. Com ele, os caracteres são formados em passos múltiplos: a primeira coluna de agulhas dispara, em seguida vem a segunda, com um ligeiro deslocamento, e assim por diante.

As impressoras gráficas permitem também a troca dos tipos de letra (conhecidos como fontes); desse modo, pode-se controlar a largura e altura dos caracteres, e determinar se eles serão impressos em negrito, em itálico, etc. (alguns programas têm comandos internos que indicam o tipo de letra a ser utilizado na impressão).

Embora todos esses métodos representem um progresso considerável, a qualidade final de impressão de textos não é irrepreensível. Para aperfeiçoá-la, é necessário utilizar as impressoras de caracteres formados, ou seja, as que têm caracteres em relevo, como as máquinas de escrever (dos tipos "esfera", "vareta", ou "margarida").

DE CARACTERES

As primeiras impressoras de caracteres formados acopladas a micros foram adaptadas de máquinas de escrever elétricas, do tipo "esfera". Embora a qualidade de impressão seja boa, a massa da cabeça de impressão (que não foi feita para as velocidades máximas de 15 cps que a impressora consegue) torna a impressão lenta e pouco confiável. Consequentemente, hoje dominam as impressoras do tipo "margarida".

As impressoras "margarida" empregam um elemento de impressão removível semelhante a uma flor, com "pétalas" irradiando do anel central. Cada "pétala" tem na ponta um caractere em relevo. Para imprimir, a impressora gira a margarida até que a "pétala" com o caractere fique em posição; um pequeno martelo pressiona-a então contra a fita de impressão e o papel. A qualidade de impressão é igual ou superior à de uma máquina de escrever elétrica, e muito superior à de uma impressora matricial (em qualidade carta); entretanto, a velocidade de impressão é bem menor, variando entre [illegible] e 80 cps.

Tais máquinas contam com recursos de ênfase de texto, como sublinhamen-

Dispostas em torno de um anel (acima), as agulhas da impressora "margarida" são equipadas com caracteres em relevo que imprimem sobre o papel (abaixo).

to, negrito etc. Além disso, elas possibilitam variações no espaçamento entre os caracteres. Algumas são capazes de gerar gráficos por meio do controle da movimentação do papel.

IMPRESSORAS TÉRMICAS

Uma desvantagem importante das impressoras matriciais e do tipo "margarida" é o custo relativamente alto. Assim, tem-se tentado, ao longo dos anos, utilizar outras técnicas mais baratas de impressão.

As primeiras tentativas levaram ao surgimento das impressoras térmicas, que exigem um papel especial, revestido com uma substância química termossensível. Esse papel é caro, mas tem a vantagem de não precisar de fita de impressão. A cabeça impressora é constituída de uma matriz equipada com pequenas peças de aquecimento. O computador seleciona o caractere a ser impresso, e o sistema eletrônico interno da impressora faz passar uma corrente elétrica nas agulhas da cabeça, aquecendo-as até cerca de 150°C. Esse padrão é então aplicado sobre o papel termossensível, escurecendo-o nos pontos correspondentes.

Tal gênero de impressora não pode ser usado para imprimir textos de melhor qualidade. O papel, geralmente de rolo, é alimentado por fricção (rolo de borracha). Assim, seu deslocamento não tem a precisão oferecida pela alimentação por trator (formulário perfurado nas margens). Entretanto, ele apresenta vantagens como: pequeno tamanho, menor desgaste mecânico etc.

Outro tipo é a chamada impressora-traçadora, cujo funcionamento se baseia no mesmo princípio do traçador digital de desenhos. Além de impressora, ela é também traçadora, utilizando pequenas canetas esferográficas de várias cores. A cabeça de impressão típica contém quatro penas, que são alojadas em um tambor rotatório. Sob comando de software, a cabeça pode ser rodada para colocar em posição a caneta com a cor adequada. Ademais, existe um solenóide que retira ou pressiona a pena contra o papel. A cabeça se movimenta no sentido transversal, e o papel no vertical, possibilitando traços contínuos sobre o papel. A velocidade de impressão se situa em torno de 12 cps. O papel é de rolo, semelhante ao das máquinas de calcular.

O PAPEL PARA IMPRESSORA

Ao selecionar uma impressora, é necessário saber antes como se quer a impressão. Isso se aplica não somente aos modelos e à qualidade dos caracteres, como também ao tipo, tamanho, cor e textura do papel. Além disso, a forma com que este é alimentado na impressora também é importante. Por exemplo, se um médico precisar imprimir receitas em papel timbrado, deverá dispor de uma impressora capaz de ser alimentada com folhas soltas.

Existe uma grande variedade de papéis para impressora de largura padronizada. Esta pode ser especificada em número de colunas (80, 120 e 132 colunas) ou ainda em centímetros (os tamanhos variam, neste caso, entre 10 e 40 cm). Os comprimentos são igualmente padronizados (ofício, A4 etc.).

Algumas impressoras usam apenas um tipo de papel (por exemplo, formulário contínuo de oitenta colunas); outras comportam vários tipos. O papel pode ser comprado em folhas soltas, em rolo ou em formulário contínuo. Este último é o papel tradicionalmente utilizado em computadores: consiste de uma faixa contínua dobrada em "Z", e acondicionada em caixas. As folhas podem ser separadas na dobra, por meio de um serrilhado ou corte vincado. Nas bor-

Além de gráficos e desenhos de todos os tipos, a impressora-traçadora pode produzir textos com caracteres de diversas cores e tamanhos.

das, existem duas fitas de papel perfurado (chamadas remalinas), que também podem ser separadas manualmente.

Cada tipo de papel exige uma forma peculiar de alimentação. Os principais mecanismos de alimentação são: trator (rodas dentadas que movimentam o formulário contínuo) e fricção (rolo de borracha, semelhante ao da máquina de escrever, usado com folhas soltas ou com papel de rolo). As folhas soltas podem ser alimentadas manualmente ou por intermédio de um dispositivo especial (este deve ser comprado separadamente, ■ não existe para todos os tipos de impressora).

Os papéis podem ser lisos (em uma só cor, geralmente branca), pautados e zebrados (com faixas de cor e brancas, alternadas). Existem ainda papéis com cópias múltiplas (1 a 7), com ou sem papel carbono intercalado.

INTERFACES

Por outro lado, ■ computador só pode se comunicar efetivamente com a impressora ■ for usada uma interface adequada.

Uma interface é, essencialmente, o hardware que torna possível a conexão entre dois sistemas. Ela é equipada com um circuito eletrônico ■ com o software necessário para operá-lo. Muitos computadores já são vendidos com ■ interface para impressora embutida; assim, basta adquirir um cabo para efetuar a ligação. Outros são normalmente vendidos sem qualquer tipo de interface para impressoras.

Quando ■ computador e a impressora são do mesmo fabricante, há *geralmente* compatibilidade entre eles. Caso isso não aconteça, eles podem ser transformados por meio de uma interface.

É importante também assegurar que o software que se pretende utilizar seja compatível com a impressora, principalmente se ele precisar de recursos especiais (elaboração de gráficos, processamento de textos etc.), peculiares a determinados tipos de impressora. Alguns softwares têm um módulo de instalação que permite selecionar a impressora a partir de um menu.

Para assegurar algum grau de compatibilidade entre os equipamentos no mercado, foram desenvolvidos diversos padrões interfaces. Os dois mais famosos são o RS-232C (uma interface do tipo serial), e o padrão Centronics (uma interface do tipo paralelo).

Numa interface serial, ■ transmissão dos bits do código de cada caractere é feita gradualmente (um de cada vez). Internamente, porém, tanto o computador quanto a impressora armazenam os bits em paralelo; assim, torna-se necessária a presença de um mecanismo de conversão de paralelo para serial (e vice-versa) em cada ponta do sistema. Daí a vantagem da interface paralela, que manda todos os bits de cada caractere ao mesmo tempo. Apesar de mais rápida, essa interface tem a desvantagem de exigir um cabo com maior número de fios (geralmente coaxial ou paralelo), enquanto ■ interface serial requer apenas um par de fios.

A conversão, transmissão e recepção de bits em série são efetuadas por um circuito integrado especial chamado UART (*Universal Assynchronous Transmitter and Receiver*, ou Transmissor e Receptor Assincrônico Universal).

Um UART no computador recebe códigos ASCII da memória por meio da UCP, "traduzindo-os" para o formato serial. Além disso, adiciona outros códigos de transmissão e os transmite para a impressora. Um UART idêntico na impressora recebe esses códigos e, se não houver erros, os converte de volta no formato paralelo. As taxas de transferência de dados (medidos em bauds) variam, segundo a impressora, entre 75 e 19 200 bits por segundo (ou bauds), o que equivale a cerca de 7 e 1 800 cps, respectivamente.

Para superar as limitações de velocidade impostas ao computador, devido à lentidão do mecanismo impressor, a maioria das impressoras dispõe de uma memória intermediária chamada *buffer*. A informação a ser impressa é enviada diretamente para o buffer, liberando o computador para outras tarefas. O controlador da impressora esvazia então o buffer na impressão. O tamanho dessa memória intermediária varia entre 8 e 8 000 bytes. À medida que ela aumenta, torna-se menor a freqüência com que o computador é interrompido, ou seja, diminui a demanda da impressora sobre o computador. Assim, embora saia mais caro, é preferível ter uma impressora com um buffer maior. Algumas empresas vendem buffers especiais de grande capacidade para serem instalados entre o computador e ■ impressora.

Um outro tipo de interface serial empregado durante muito tempo com impressoras é o laço de corrente de 20 mA. Originalmente usado em terminais de teletipos e telex, esse tipo tem sido substituído ultimamente pelo padrão RS-232C, pois é muito lento.

A interface paralela mais usada é a do tipo Centronics. Desenvolvida pelo fabricante do mesmo nome na década de 70, ela foi universalmente adotada devido à sua simplicidade. A esmagadora maioria de impressoras e micros utiliza hoje esse padrão, o que garante ampla compatibilidade.

Outra interface bastante usada, principalmente ■ interligação de instrumentos de medida a computadores, é o padrão IEE 488, um sofisticado equipamento paralelo que permite a operação full-duplex (comunicação simultânea nos dois sentidos). Como pode ser conectada a muitos micros, essa interface foi incorporada a alguns modelos de impressora. Muitos de seus inúmeros recursos, porém, não são necessários à comunicação computador-impressora. Por esse motivo, a Centronics continua sendo a opção preferida.

CONJUNTOS DE BLOCOS GRÁFICOS (1)

Monstros ■ espaçonaves são apenas duas aplicações dos blocos gráficos. Extremamente úteis ■ versáteis, estes podem assumir a forma que quisermos. A imaginação ■ o limite.

Nem só de monstros e dragões vivem os blocos gráficos. Com eles podemos criar, por exemplo, conjuntos de símbolos especiais ou de novas letras, ou qualquer outra coisa que desejarmos. Neste artigo, veremos como criar uma grande variedade de blocos e como utilizar melhor o programa gerador apresentado nas lições anteriores.

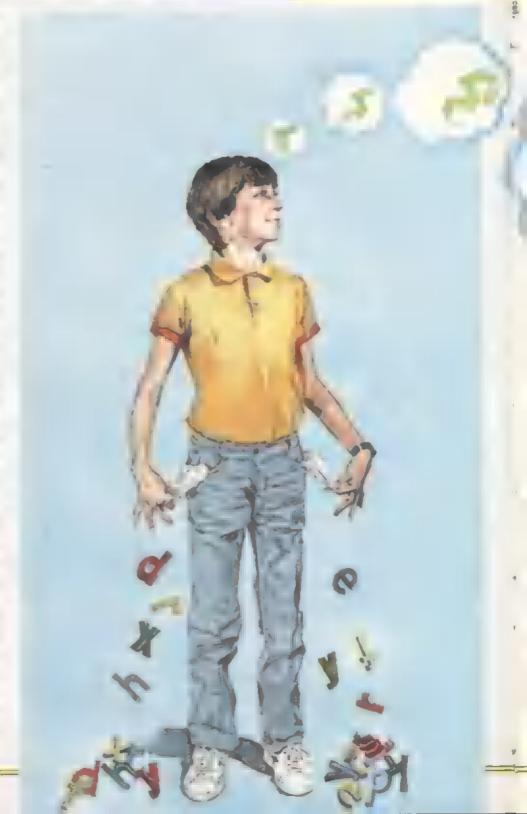
QUAIS SÃO OS LIMITES?

Alguns microcomputadores limitam ■ número de blocos que podem ser produzidos. Este é o caso do Spectrum, que só permite ■ criação de 21 blocos. Já no MSX é possível produzir 256 blocos para cada terço da tela de alta resolução — 256 também é o máximo que o programa gerador cria de cada vez. No Apple, ■ número de blocos é limitado apenas pela quantidade de memória disponível; esse é também o limite do TRS-Color. Neste último, aliás, não se pode falar propriamente de blocos gráficos, mas sim de matrizes que armazenam o padrão desejado. O ZX-81 ■ o TRS-80 não dispõem desse recurso, impossibilitando ■ criação de blocos gráficos em BASIC apenas.

No caso do Spectrum, há formas de ■ extrapolar o limite dos 21 blocos, quando se deseja um número maior. Assim, podemos criar uma tela que use muitos blocos — por exemplo, um jogo em que ■ sapo deve atravessar uma pista movimentada, pela qual trafegam carros, caminhões e ônibus. Isso certamente consumirá mais de 21 caracteres, sem contar os que vamos precisar para desenhos de fundo como margens de rios, calçadas ■ acostamentos.

Suponhamos que se trate de um desenho representando uma rua. Teremos, nesse caso, de utilizar blocos para figurar os edifícios, as pessoas, os veículos e outros detalhes da cena. No próximo artigo, explicaremos como criar uma tela desse tipo.

Finalmente, podemos precisar de um novo conjunto de caracteres — o alfabeto grego, por exemplo — ■ que ultrapassaria o limite de 21 caracteres.

Em resumo, há inúmeras situações que exigem mais de 21 blocos gráficos.

- ■ QUANTOS UDG PODEMOS CRIAR?
- ■ RESERVE UM ESPAÇO NA MEMÓRIA PARA OS BLOCOS
- ■ ESCREVA NA TELA DE ALTA RESOLUÇÃO DO APPLE
- ■ UM NOVO CONJUNTO DE CARACTERES NO SPECTRUM E NO TK-2000
- ■ A ORGANIZAÇÃO DA MEMÓRIA DE ALTA RESOLUÇÃO DO MSX

Felizmente, casos como esses não ficam sem solução, pois existe uma maneira de aumentar o número de blocos oferecidos pelo computador.

O espaço reservado pelo Spectrum ao ser ligado corresponde a 21 blocos gráficos. Podemos, contudo, aumentá-lo, destinando uma área maior na RAM especialmente para esse fim. Assim, quando os padrões dos novos blocos estiverem na memória, serão formados vários "bancos" de 21 blocos cada um.

COMO COLOCAR OS BLOCOS NA MEMÓRIA

A primeira coisa a fazer é encontrar um lugar seguro para guardar nossos blocos na memória, de tal forma que eles não possam ser apagados pelo BASIC. Devemos também decidir quantos blocos usaremos para dimensionar previamente o espaço ■ ser reservado.

Se quisermos criar mais 21 blocos, podemos fazer ■ cálculo do comprimento que o "banco de memória" vai ocupar, multiplicando 21 (o número de novos blocos) por 8, que é o número de bytes necessários para definir o padrão de um bloco. O resultado dessa multiplicação é o número de bytes que precisamos reservar.

O passo seguinte consiste em escolher a posição em que vamos colocar o "banco". Quanto mais alto o colocarmos na memória, mais espaço sobrará para o programa em BASIC. O ideal seria posicionar nosso "banco" imediatamente abaixo da área em que ficam guardados os 21 primeiros blocos e reservada automaticamente pelo micro.

O endereço máximo até onde o BASIC pode chegar é denominado RAMTOP. Ele fica situado uma posição abaixo do primeiro byte da área dos blocos gráficos — UDG —, reservada automaticamente pelo Spectrum. Veremos mais tarde que esse endereço não é fixo. Podemos verificar o valor atual de RAMTOP por intermédio do comando:

```
PRINT USR "A" - 1
```

Essa linha mostra na tela o endereço do primeiro byte do bloco definido pelo usuário no lugar "A" menos 1. Se não houver alterações depois que o aparelho for ligado, o valor de RAMTOP deverá ser 65367 no Spectrum de 48K, e 32599 no de 16K.

Agora que sabemos onde fica o RAMTOP, podemos calcular o endereço inicial de nosso espaço reservado para os novos blocos.

Para isso, temos que calcular quantos bytes vamos reservar a partir do primeiro byte da área dos UDG. Nosso banco tem um comprimento de 21 x 8, ou 168 bytes. Dessa forma, ele deve começar 168 bytes abaixo da área dos UDG. Isso fica na posição:

```
PRINT USR "A" - 169
```

que corresponde ao valor 65199, ou 32431 nas máquinas de 16K.

PROTEÇÃO DO BANCO DE BLOCOS

Entretanto, não basta colocar simplesmente os padrões dos blocos nessa área. Programas BASIC muito longos podem chegar até lá, destruindo os dados. Para proteger essa área, devemos

digitar, ou colocar em uma linha do programa, ■ instrução:

```
CLEAR USR "A" - 169
```

Isso fará com que ■ RAMTOP seja deslocado 168 bytes para baixo, de modo que as posições acima de 65199 (ou 32431) ficarão protegidas.

Agora, temos que informar o Spectrum onde encontrar os dados; depois disso, colocamos na memória os bytes correspondentes aos novos caracteres.

COMO USAR APONTADORES

Para informar ao computador onde se encontram os dados referentes aos novos blocos, precisamos modificar o valor de um *apontador*.

Existem vários apontadores ■■ memória do Spectrum: cada um deles "aponta" para uma posição importante da memória. Neste caso específico, estamos interessados naquele que "aponta" para o endereço do primeiro byte da área de blocos gráficos. Antes de usar um bloco gráfico, o computador deve descobrir onde seus bytes estão guardados. Para isso, ele precisa primeiro obter o valor do endereço que está no apontador. Esse valor é geralmente 65368, mas devemos modificá-lo, de maneira que ele passe a "apontar" para o início do novo banco de blocos.

Uma característica dos apontadores é que podemos colocar novos valores dentro deles usando **POKE**; assim, o computador utilizará simplesmente o novo endereço. É essa característica que faz do apontador um elemento tão útil na expansão do limite de blocos. Na verdade, precisamos de dois comandos **POKE** para modificar o conteúdo de um apontador, pois ■ valor máximo que um byte pode conter é 255, ao passo que os valores de endereços são geralmente maiores. Desse modo, o Spectrum quebra o endereço em duas partes, equivalentes, por assim dizer, ao algarismo das dezenas e ao algarismo das unidades de um número normal.

Para calcular as diferentes partes de um número decimal, devemos dividi-lo por 10, caso ele tenha dois algarismos. Tomemos o número 56 como exemplo: para encontrar o algarismo das dezenas, dividimos 56 por 10 ■ obtemos 5. O resto da divisão, 6 no nosso exemplo, é o algarismo das unidades.

Quando quisermos colocar um novo valor dentro do apontador (usando **POKE**), devemos quebrar o número em duas partes de maneira similar. A diferença é que, em vez de dividirmos por 10, dividimos por 256. Nosso banco contendo 21 novos blocos deve começar no endereço 65200 (no Spectrum de 48K). Assim, é preciso colocar esse valor dentro do apontador, quebrando-o em duas partes. Dividimos então 65200 por 256. Obtemos 254 ■ um resto igual ■ 176.

O sistema operacional do Spectrum interpreta a primeira parte de um número com duas partes como sendo a menor (ao contrário do que fazemos com números decimais). Desse modo, o quociente da divisão vai para a segunda parte do número ■ o resto para a primeira.

O nosso apontador corresponde então a dois endereços: 23675 e 23676. Portanto, precisamos de dois **POKE**:

```
POKE 23675,176
POKE 23676,254
```

Falta agora criar e colocar os novos blocos na memória.

Uma vez modificado o conteúdo do apontador, procedemos exatamente da mesma maneira que com blocos normais: **POKE USR "A"** com o primeiro byte do primeiro caractere, **USR "A"** + **1** com o segundo byte do primeiro caractere, e assim por diante.

Se você preferir, pode colocar os valores na memória antes de mudar ■ apontador. Neste caso, terá que usar o valor do endereço, e não mais **USR "A"**. Assim, no nosso exemplo, o primeiro endereço seria 65200. Para não confundirmos ■ números, é aconselhável usar + **1** em cada novo **POKE**.

Para trabalhar com os caracteres novos, é preciso modificar ■ apontador (geralmente, é mais fácil fazer isso no início).

Se seguirmos corretamente as instruções, será fácil imprimir os blocos, usando um programa BASIC. Contudo, quando precisamos de novos caracteres ou blocos disponíveis via teclado, o método não é muito adequado. Não é agradável mudar o apontador ou usar o modo gráfico ■ todo instante.

UM NOVO CONJUNTO DE CARACTERES

Para contornar ■ problema, é necessário que redefinamos o conjunto de

caracteres do Spectrum. Isso vai fazer com que o ato de pressionar uma tecla imprima um novo caractere ou bloco gráfico, em vez da letra ou número correspondentes.

Há muitos motivos para se criar um novo conjunto de caracteres: por exemplo, quando queremos imprimir mensagens na tela em outra língua (russo ou grego, digamos), ou quando desejamos personalizar um programa, usando tipos que nós mesmos criamos.

A redefinição do conjunto de caracteres do Spectrum é na realidade bem similar à criação de novos bancos de UDG: reserva-se uma área na memória RAM e coloca-se ali os padrões de maneira análoga.

O único problema com a redefinição é que, mesmo que queiramos criar apenas um caractere novo, todos os outros devem ter seus padrões transferidos para a mesma área da RAM.

A razão para isso é que devemos modificar o apontador do conjunto de caracteres, da mesma maneira que anteriormente mudamos o apontador de UDG. Uma vez modificado o valor do apontador, o computador passa a procurar os padrões das letras no novo endereço na RAM, não podendo retornar à ROM para obter os padrões das letras que não foram modificadas.

Um procedimento útil é transferir para a área reservada na RAM todos os bytes das letras antes de colocar os dados correspondentes aos caracteres que queremos modificar.

Digite a instrução **RANDOM O** para limpar a memória e depois carregue o seguinte programa:

```
 10 CLEAR USR "A"-769
 20 LET d=PEEK 23730+256*PEEK
 23731+1
 30 FOR n=15616 TO 15616+767
 40 POKE d,PEEK n
 50 LET d=d+1: NEXT n
 60 POKE 23606,PEEK 23675
 70 POKE 23607,PEEK 23676-4
 80 LET p=PEEK 23606+256*PEEK
 23607+48*8
 90 FOR n=p TO p+79: READ a:
 POKE n,a: NEXT n
110 DATA 0,124,76,84,86,102,
126,0
130 DATA 0,8,8,8,24,24,24,0
150 DATA 0,126,2,126,96,96,126
,0
170 DATA 0,124,4,126,6,6,126,0
190 DATA 0,96,98,98,126,2,2,0
210 DATA 0,124,64,126,6,6,126,0,
230 DATA 0,62,32,126,98,98,126
,0
250 DATA 0,124,4,4,6,6,6,0
270 DATA 0,60,36,60,102,102,
126,0
290 DATA 0,124,68,126,6,6,126,
0
```

As oito primeiras linhas protegem uma área na RAM e colocam ali os bytes correspondentes ao conjunto de caracteres obtidos na ROM. As linhas restantes redefinem os números.

Coloque o programa para funcionar, aguardando a mensagem "OK". A seguir, aperte uma das teclas numéricas.

Se pressionarmos **NEW**, os números voltarão à sua forma original. Entretanto, os dados correspondentes continuam na memória e o novo conjunto de caracteres pode ser reativado por:

```
POKE 23606, PEEK 23675
POKE 23607, PEEK 23676 - 4
```

Não importa o tamanho da memória de seu micro — 16K ou 48K —, o programa descobre e altera os endereços relevantes. Ele faz isto verificando o valor do apontador do **RAMTOP** (endereços 23730 e 23731). Os valores dos comandos **PEEK** são combinados de maneira a produzir o valor de **RAMTOP**.

O conjunto de caracteres da ROM é guardado entre os endereços 15616 e 15616 + 767 (pelo menos, esta é a parte do conjunto de caracteres que pode ser redefinida). Usando um laço **FOR...NEXT**, o programa transfere para a RAM os bytes dessa região da ROM.

O laço atualiza o endereço que está sendo transferido automaticamente (já que o contador do laço é também o endereço que está sendo transferido); o endereço da RAM que está recebendo o byte é atualizado na linha 50 (variável **d**). Como podemos ver na linha 30, não há necessidade de somar os endereços nem os bytes, já que o Spectrum pode fazer isso por nós. As linhas 60 e 70 modificam o valor do apontador do conjunto de caracteres, para que ele "aponte" para o endereço inicial da área que reservamos na RAM. Esse endereço é calculado a partir do endereço da área de caracteres UDG, obtido por intermédio de **PEEK** (esse mecanismo funcionará mesmo que tenham sido criados novos bancos de UDG). O novo conjunto de caracteres ficará logo abaixo da área de caracteres UDG; assim, o cálculo na linha 70 simplesmente subtrai o comprimento do conjunto de caracteres do endereço inicial da área de UDG. Esse comprimento é igual a 4, mas, como está no byte mais significativo do endereço, ele é multiplicado por 256, resultando no comprimento real do conjunto, que é de 1024 bytes.

Agora que o conjunto de caracteres foi todo transferido para a RAM, podemos substituir alguns dos velhos caracteres por novos.

Se não tomarmos cuidado, corremos o risco de substituir os caracteres errados. Isso pode ser evitado se escolhermos direito os caracteres que serão modificados e se colocarmos os novos bytes nos lugares certos.

O apêndice 1 do manual do Spectrum apresenta uma lista com o conjunto de caracteres. Somente podem ser redefinidos os caracteres com códigos ASCII entre 32 e 127 (a coluna da esquerda traz o código de cada caractere).

Para calcular os bytes que devem ser modificados, multiplicamos o código do caractere por 8. Essa operação nos dá o endereço do primeiro byte do caractere que queremos redesenhar. Para o caractere de espaço (código ASCII 32), o resultado é 32 × 8 = 256.

Agora que sabemos onde o byte está, dentro do conjunto de caracteres, somamos esse número ao endereço que se encontra no apontador do conjunto de caracteres. Isso é calculado como **PEEK 23606 + 256*PEEK 23607 + 32*8**. No Spectrum de 48K recém-ligado, o resultado será 64600.

Depois de tudo isso, torna-se fácil fornecer os dados do novo caractere ao Spectrum. Usamos **POKE** para colocar oito bytes na memória a partir do endereço que acabamos de calcular. Esses bytes devem corresponder ao padrão do novo bloco.

As linhas que se seguem redefinem o formato dos caracteres de espaço, fazendo com que eles se tornem visíveis na listagem:

```
10 FOR X=64600 TO 64600+7
20 READ A
30 POKE X,A
40 NEXT X
50 DATA 0,126,66,66,66,66,126
,0
```

Se seu micro tem 16K, mude o número 64600 para 31832. Evidentemente, é uma boa idéia usar laços **FOR...NEXT**

para colocar bytes na memória, uma vez que o laço muda automaticamente o endereço e evita que escrevamos oito comandos **POKE**.

Tente agora calcular o endereço inicial dos bytes correspondentes aos diversos caracteres numéricos: você poderá conferir o resultado examinando a listagem do programa.

Já vimos como mudar o formato dos caracteres do MSX. Agora, vamos procurar esclarecer a organização da memória de alta resolução (**SCREEN 2**) para que possamos criar blocos gráficos (à mão ou por meio do gerador de blocos). O artigo *Os Comandos PEEK e POKE* (página 261) apresenta uma introdução ao assunto, que deve ser revista.

A tela do MSX tem 256 pontos de largura por 192 de altura. Cada um desses pontos pode estar aceso ou apagado, assumindo assim a cor de frente ou a cor de fundo, respectivamente. Existem dezesseis cores disponíveis. No modo de alta resolução (**SCREEN 2**), esses pontos estão distribuídos por 768 blocos gráficos de 8 x 8 pontos. Cada bloco gráfico definido pelo usuário pode ocupar uma dessas 768 posições — ao contrário dos sprites, que podem ocupar qualquer posição.

O padrão de cada uma dessas posições (ou blocos) é definido da maneira usual, ou seja, cada linha de oito pontos corresponde a um número de oito bits ou um byte. Assim, cada posição precisa de oito bytes de memória.

O MSX tem uma memória separada só para o vídeo: a memória VRAM. O comando **SCREEN** determina como essa memória será utilizada pelo micro. No modo de alta resolução — que obtemos logo após acionar o comando **SCREEN 2** dentro de um programa BASIC —, uma porção da VRAM é separada para armazenar os padrões de todas as 768 posições da tela. Essa parte da VRAM é denominada *tabela de padrões* e tem seu endereço inicial (na VRAM) contido em **BASE(12)**, que é uma variável interna do micro. Como cada posição — 8 x 8 pontos — da tela necessita de oito bytes, a tabela de padrões tem um comprimento igual a 768 x 8 = 6144 bytes. Cada byte define o padrão de uma linha de bloco gráfico.

Esse padrão, por sua vez, determina quais pontos estão acesos e quais pontos estão apagados. Para colorir os pontos, o computador precisa usar uma outra área da VRAM, a *tabela de cores*, que tem endereço inicial armazenado em **BASE(11)**. Cada linha de bloco gráfico pode ter apenas uma cor de frente (pontos acesos) e uma de fundo (pontos apagados). Assim, a cada linha de bloco gráfico — ou posição — da tela corresponde um byte de cor. O valor desse byte é igual ao código da cor de fundo mais 16 multiplicado pelo código da cor de frente. Como a tela tem 768 posições e cada posição oito linhas, a tabela de cores também tem 6144 bytes de comprimento.

Quando quisermos colocar um bloco gráfico em determinado ponto da tela, devemos calcular as posições nas tabelas de padrões e de cores correspondentes aos bytes que vamos alterar. Para conhecermos a posição na tabela de cor ou na de padrões do primeiro byte que coloca um bloco na tela, multiplicamos a posição desejada na tela por oito. Se quisermos colocar um bloco, digamos, na posição 25 da tela, temos que alterar oito bytes, a partir da posição 25 x 8 = 200 na tabela de padrões e na de cores. O programa a seguir tenta ilustrar o processo:

```
5 P=367
10 SCREEN 2
20 FOR I=0 TO 7
30 READ A(I)
40 VPOKE BASE(12)+P*8+I,A(I)
50 VPOKE BASE(11)+P*8+I,6*16+11
60 NEXT I
70 GOTO 70
100 DATA 254,124,56,16,8,28,62,
127
```

Este programa desenha na posição 367 da tela o bloco gráfico cujo padrão corresponde aos oito bytes da linha **DATA** 100.

A linha 5 determina a posição P da tela onde será colocado o bloco gráfico. A linha 10 seleciona o modo de alta resolução.

O comando **VPOKE** é utilizado para colocar valores na VRAM. Note como os endereços correspondentes ao padrão e à cor das linhas do bloco são calculados. Como são oito linhas, o laço **FOR...NEXT** será repetido oito vezes. Na linha 40, o endereço do primeiro byte a ser alterado na tabela de padrões é 8 multiplicado pela posição na tela P. Essa posição é contada na tabela a partir de seu endereço inicial na VRAM, dado por **BASE(12)**. A variável **I**, contador do laço, faz com que oito posições consecutivas da VRAM sejam modificadas. Por fim, o valor colocado na tabela de padrões é A(I), que foi lido com **READ** na linha 30. A cor de cada linha do bloco é determinada de forma semelhante. O cálculo da posição é o mesmo, só que o valor obtido é contado a partir de **BASE(11)**, endereço inicial da tabela de cores na VRAM. O byte de cor determina que a cor de frente seja vermelha (código 6) e a de fundo seja amarela (código 11), na linha 50.

A linha 70, por sua vez, evita que a tela seja apagada, já que o modo gráfico de alta resolução — **SCREEN 2** — só permanece ativo enquanto o programa estiver funcionando.

O PROGRAMA GERADOR DE BLOCOS

Se você já criou vários blocos gráficos com o auxílio de nosso programa gerador, chegou o momento de usá-los. Se você ainda não usou o programa, o que vem a seguir pode parecer confuso. Assim, para aproveitar bem as próximas linhas, é necessário que você tenha criado e guardado em fita todo um banco cheio de blocos gráficos criados pelo gerador.

Uma vez gravado um banco contendo 256 blocos, podemos recuperá-lo de duas maneiras: usando o próprio programa gerador para fazer uma edição, por exemplo, ou para retirá-lo diretamente da memória do micro e usar os blocos em outro programa. Na segunda hipótese, a primeira coisa a fazer é proteger uma área no topo da memória do micro, colocando ali o banco de blocos. Para isso, digite:

```
CLEAR 200,&HC999
```

Em seguida, posicione a fita e carregue o banco de blocos usando:

```
BLOAD "CAS:"
```

e o banco terá sido carregado.

Para trazer o banco para a tela, use um programa BASIC:

```
5 SCREEN 2
10 FOR J=0 TO 2*256*8 STEP 256*
8
20 FOR I=0 TO 256*8-1
```

```
30 VPOKE BASE(12)+I+J,PEEK(&HD1
00+I)
40 VPOKE BASE(11)+I+J,PEEK(&HE1
00+I)
50 NEXT I,J
60 GOTO 60
```

Depois de executar o programa, teremos três cópias do banco de blocos que criamos anteriormente. Todos os blocos produzidos estarão lá. Se não tivermos armado um banco completo com 256 blocos, alguns blocos estranhos aparecerão, mas sem prejuízo dos que foram criados. Esses blocos exteriores correspondem a valores da memória que permaneceram inalterados quando criamos o banco e foram gravados junto com ele. No entanto, nada disso acontecerá se criarmos 256 blocos, preenchendo inteiramente o espaço.

A linha 5 ativa a tela de alta resolução. A linha 10 se prepara para fazer três cópias do banco, repetindo três vezes as linhas de 20 a 40. O laço entre as linhas 30 e 40 será repetido 256 x 8 vezes — correspondente a 256 blocos com oito bytes cada um.

A linha 30 transfere os padrões do banco — endereço inicial D100, em hexa — para a tabela de padrões — endereço inicial **BASE(12)**. A linha 40 transfere os bytes de cor do banco — endereço inicial E100, em hexa — para a tabela de cores — endereço inicial **BASE(11)**. Observe como esse processo é demorado.

Como vemos, o banco de blocos criado pelo gerador consiste em uma "tabela de padrões" — endereço inicial D100, hexadecimal — e uma "tabela de cores" — endereço inicial E100.

Obviamente, nem sempre desejamos transferir de uma só vez todo o banco para a tela, como no exemplo. Quando quisermos obter o padrão (ou a cor) de um bloco isolado que está no banco, teremos que calcular onde se encontram os bytes correspondentes. Para isso, precisamos saber qual a posição desse bloco no banco (é necessário fazer uma lista quando criarmos os blocos). Conhecendo esse valor, é possível calcular o endereço do primeiro byte do bloco, multiplicando sua posição por 8 e somando o resultado ao endereço inicial — D100 ou E100, conforme queiramos padrões ou cores.

A TABELA DE NOMES

Por este método de trabalho com a tela de alta resolução, tudo o que foi colocado nas tabelas de padrões e cores apareceu imediatamente na tela. Existe, porém, outra maneira de usar essas tabelas: levando-as a funcionar como bancos de blocos gráficos, dos quais escolhemos apenas alguns para exibir de cada vez na tela.

Para fazer isso, utilizamos a tabela de nomes — endereço inicial BASE (10), na VRAM. Essa tabela tem 768 bytes de comprimento — cada byte corresponde a uma posição (8 x 8 pontos) da tela. Ela permite que codifiquemos os blocos que estão na tabela de padrões com números de 0 a 255, como no banco do gerador. Para que o bloco de número 37, por exemplo, apareça na posição 170 da tela, basta fazer com que o byte 170 da tabela de nomes seja igual a 37, usando **VPOKE BASE(10) + 170,37**. Essa organização é semelhante à da tela de textos, onde os padrões dos caracteres ficam numa tabela de padrões e uma tabela de nomes faz com que os caracteres sejam impressos na tela codificados por seu código ASCII.

Existem aqui dois problemas. Primeiro, só podem ser codificados 256 blocos gráficos, enquanto o número de blocos nas tabelas de cor e padrão é três vezes maior. Esse problema é contornado com a criação de três bancos de blocos gráficos. Isso significa que os 256 primeiros blocos (256 x 8 bytes) das tabelas de cor e padrão têm seus códigos válidos apenas para o terço superior da tela. Do mesmo modo, o terço médio e o inferior contam com seus próprios bancos — respectivamente, no terço médio e inferior das tabelas de cor e padrão.

O segundo problema é que nem o próprio micro emprega esse tipo de organização. Assim, se trabalharmos com a codificação descrita para criar telas gráficas, não poderemos recorrer a comandos como **DRAW**, **LINE** e **PAINT**. A forma de organização utilizada pelo computador é a mesma que faz com que cada byte colocado na tabela de padrões e de cores apareça na tela, possibilitando o modo de utilização que descrevemos linhas atrás.

Para entender melhor, veja os valores que o micro mantém normalmente na tabela de nomes:

```
20 FOR I=0 TO 3*256-1
30 PRINT I,VPEEK(BASE(10)+I)
40 NEXT
```

Este programa lista os valores contidos na tabela de nomes. Note que a posição 1 corresponde ao caractere 1, que é o primeiro da tabela de padrões e de cores; a posição 2 tem valor 2, correspondente ao segundo bloco da tabela de padrões (e de cores), e assim por diante.

Para familiarizar-se com a utilização da tabela de nomes, acrescente as próximas linhas ao programa que carrega o banco criado pelo gerador. Ele só funcionará, contudo, se houver um banco de blocos no topo da memória.

```
60 FOR I=0 TO 255
70 FOR J=0 TO 3*256-1
80 VPOKE BASE(10)+J,I
90 NEXT J,I
100 GOTO 100
```

Este programa preenche toda a tabela

A figura mostra na linha superior os números empregados pelo Spectrum; logo abaixo, aparecem alinhados os novos números criados pelo programa.

de nomes com o mesmo valor. O código de cada bloco é modificado pelo laço **FOR...NEXT** das linhas 60 a 90, de tal forma que, ao ser desenhado, cada um dos 256 blocos do banco ocupa a tela inteira.

Mais interessante ainda é fazer com que os blocos que estão na tela se movimentem. Para tanto, você deve modificar a linha 80:

```
80 VPOKE BASE(10)+J,(VPEEK(BASE
(10)+J)+1)AND 255
```

O programa movimenta todos os blocos da tela mudando apenas os valores na tabela de nomes. Uma vez copiado o banco do alto da memória nas tabelas de cor [illegible] padrão, seus valores não se modificam. A velocidade com que são movimentados os blocos é aproximadamente dezesseis vezes maior que aquela com que eles são desenhados no início do programa, já que um **VPOKE** na tabela de nomes equivale a oito **VPOKE** na tabela de padrões [illegible] oito **VPOKE** na tabela de cor.

Esta última modificação faz com que apenas [illegible] terço superior da tela seja movimentado:

```
70 FOR J=0 TO 255
```

O maior problema para a utilização de blocos gráficos no Apple e no TK-2000 é a caótica organização da memória de vídeo desses computadores. Com efeito, na tela desses micros, as linhas consecutivas não têm números consecutivos e é necessário recorrer à matemática para contornar a dificuldade, o que torna parte do programa muito complicada para alguns.

Nada melhor do que um exemplo em BASIC para ilustrar o problema:

```
10  HGR
20  FOR I = 8192 TO 16383
30  POKE I,255
40  NEXT I
```

Este programa coloca na tela pedaços de linha, por intermédio de **POKE**. Embora os endereços da memória de vídeo sejam consecutivos, as linhas produzidas não o são. Ora, nossos blocos gráficos só serão desenhados se colocarmos valores na memória de vídeo usando **POKE**.

O problema pode ser resolvido pela dedução de uma fórmula matemática que permita calcular os oito endereços necessários para desenhar um bloco gráfico em uma determinada posição.

[illegible] NA TELA DE ALTA RESOLUÇÃO

Para os usuários do TK-2000 o programa a seguir pode parecer desnecessário, pois esse micro é capaz de escrever textos na tela de alta resolução. Entretanto, os usuários do Apple o acharão bem útil.

```
10  HGR :E = 16384:T = 8192
20  FOR I = E TO E + 9 * 8 - 1:
 [illegible] B: POKE I,B: NEXT
30  DATA  28,34,38,42,50,34,28,
0
40  DATA  16,24,20,16,16,16,56,
0
50  DATA  28,34,32,24,4,2,62,0
60  DATA  28,34,32,24,32,34,28,
0
70  DATA  16,24,20,18,62,16,16,
0
80  DATA  62,2,2,30,32,34,28,0
90  DATA  28,34,2,30,34,34,28,0
100  DATA  62,32,32,16,8,8,8,0
110  DATA  28,34,34,28,34,34,28
,0
120  DATA  28,34,34,60,32,34,28
,0
130  FOR Y = 0 TO 19: FOR X = 0
 TO 39:N =  INT ( RND (1) [illegible] 9)
140  FOR I = 0 TO 7
150  POKE T + (Y - 8 * (Y > 7)
- 8 [illegible] (Y > 15)) [illegible] 128 + 40 * (Y
 > 7) + 40 * (Y > 15) + X + 102
[illegible] * I, PEEK (E + [illegible] * [illegible] + I)
160  NEXT I,X,Y
```

O programa começa ativando a tela de alta resolução e estabelecendo os valores de E, endereço inicial da página 2, e T, endereço inicial da página 1. Os usuários do TK-2000 devem mudar o valor de T para 40960.

A linha 20 lê os valores contidos nas linhas **DATA**, criando parte de um banco de blocos.

A linha 130 inicia um laço **FOR...NEXT**, onde Y é a linha [illegible] X a coluna em que serão impressos os números. N é o número que será impresso, escolhido ao acaso.

A parte mais importante do programa [illegible] **FOR...NEXT** das linhas 140 a 160. Ela é responsável pelos oito comandos **POKE** que colocam o bloco na posição desejada. A fórmula a que nos referimos é a que está na linha 150.

Este programa só escreve números. Ora, nosso objetivo é escrever todo tipo de mensagem na tela gráfica. Para isso, precisamos criar (com o auxílio do gerador) um conjunto inteiro de caracteres. Quando houver um banco de blocos com caracteres na página 2, poderemos utilizar seus bytes para escrever na tela de alta resolução.

Os usuários do TK-2000 devem agora redefinir o conjunto de caracteres, criando seus próprios tipos gráficos.

Para os que não se habituaram [illegible] uso do gerador, o programa a seguir cria um banco de blocos contendo os caracteres de código ASCII entre 32 e 95.

Uma vez executado com sucesso, ele poderá ser apagado e todo o banco será visualizado e editado pelo gerador de blocos. Basta carregá-lo sem desligar [illegible] computador.

As letras têm aqui posições correspondentes a seus códigos ASCII.

```
10  HGR :E = 16384:T = 8192
20  FOR I = E +.32 [illegible] 8 TO [illegible] + 3
[illegible] * [illegible] + 64 * [illegible] - 1: READ B: POK
[illegible] I,B: NEXT
100  DATA  0,0,0,0,0,0,0,0
110  DATA  8,8,8,8,0,0,8,0
120  DATA  18,18,18,0,0,0,0,0
130  DATA  18,18,63,18,63,18,18
,0
140  DATA  8 ,60 ,10 ,28 ,40 ,3
0 ,8 ,0
150  DATA  38,38,16,8,4,50,50,0
160  DATA  12,18,18,12,82,34,92
,0
```

```
170  DATA  24,16,8,0,0,0,0,0
180  DATA  32,16,8,8,8,16,32,0
190  DATA  2,4,8,8,8,4,2,0
200  DATA  0,8,42,28,28,42,8,0
210  DATA  0,8,8,62,8,8,0,0
220  DATA  0,0,0,0,0,24,16,8
230  DATA  0,0,0,62,0,0,0,0
240  DATA  0,0,0,0,0,24,24,0
250  DATA  64,32,16,8,4,2,1,0
260  DATA  28,34,38,42,50,34,28
,0
270  DATA  16,24,20,16,16,16,56
,0
280  DATA  28,34,32,24,4,2,62,0
290  DATA  28,34,32,24,32,34,28
,0
300  DATA  16,24,20,18,62,16,16
,0
310  DATA  62,2,2,30,32,34,28,0
320  DATA  28,34,2,30,34,34,28,
0
330  DATA  62,32,32,16,8,8,8,0
340  DATA  28,34,34,28,34,34,28
,0
350  DATA  28,34,34,60,32,34,28
,0
360  DATA  0,24,24,0,0,24,24,0
370  DATA  0,24,24,0,0,24,16,8
380  DATA  32,16,8,4,8,16,32,0
390  DATA  0,0,0,62,0,62,0,0
400  DATA  4,8,16,32,16,8,4,0
410  DATA  28,34,32,16,8,8,0,8
420  DATA  28,34,58,42,58,2,60,
0
430  DATA  8,20,34,34,62,34,34,
0
440  DATA  30,34,34,30,34,34,30
,0
450  DATA  28,34,2,2,2,34,28,0
460  DATA  30,34,34,34,34,34,30
,0
470  DATA  62,2,2,62,2,2,62,0
480  DATA  62,2,2,30,2,2,2,0
490  DATA  28,34,2,58,34,34,28,
0
500  DATA  34,34,34,62,34,34,34
,0
510  DATA  24,8,8,8,8,8,28,0
520  DATA  56,16,16,16,18,18,28
,0
530  DATA  34,18,10,6,10,18,34,
0
540  DATA  2,2,2,2,2,2,62,0
550  DATA  65,99,85,73,65,65,65
,0
560  DATA  34,38,42,42,50,34,34
,0
570  DATA  28,34,34,34,34,34,28
,0
580  DATA  30,34,34,30,2,2,2,0
590  DATA  28,34,34,34,42,50,60
,64
600  DATA  30,34,34,30,18,34,34
,0
610  DATA  60,2,2,28,32,32,30,0
620  DATA  62,8,8,8,8,8,8,0
630  DATA  34,34,34,34,34,34,28
,0
640  DATA  34,34,34,34,34,20,8,
0
650  DATA  65,65,65,73,85,99,65
,0
660  DATA  34,34,20,8,20,34,34,
0
670  DATA  34,34,34,20,8,8,8,0
680  DATA  62,32,16,8,4,2,62,0
690  DATA  60,4,4,4,4,4,60,0
700  DATA  1,2,4,8,16,32,64,0
710  DATA  30,16,16,16,16,16,30
,0
720  DATA  8,20,34,0,0,0,0,0
730  DATA  0,0,0,0,0,0,62,0
799  HGR
800 A$ = "ESTA MENSAGEM E' [illegible] T
ESTE"
810 C = 7:L = 11: [illegible] 1000
820  END
1000 N = L * 40 + C
1010  FOR J = 1 TO  LEN (A$)
1020 B$ =  MID$ (A$,J,1)
1030 B =  ASC (B$)
1040 L =  INT (N / 40):C = N -
40 * L
1050  FOR I = 0 TO 7
1060  POKE T + (L - 8 * (L > 7)
 - 8 * (L > 15)) * 128 + 40 * (
L > 7) + 40 * (L > 15) + C + 10
24 * I, PEEK (E + B * 8 + I)
1070  NEXT I:N = N + 1: NEXT J
1080  HCOLOR= 6
1090  HPLOT 25,80 TO 250,80 TO
250,105 TO 26,105 TO 26,80
1100  RETURN
```

A linha 10 estabelece as condições iniciais. A linha 20 cuida de transferir para a página 2 de vídeo os padrões dos caracteres contidos nas linhas **DATA**. Note que são 64 caracteres, a partir da posição 32, e cada caractere precisa de oito bytes para ser definido. Com essas informações fica mais fácil entender os números da linha.

Para imprimir uma mensagem, utiliza-se a sub-rotina que começa na linha 1000. Essa sub-rotina imprime na tela gráfica o conteúdo da variável alfanumérica **A$**. Para fazer isso, ela toma letra por letra da variável, usando **MID$** na linha 1020. Como o código ASCII da letra é igual à sua posição no banco, fica fácil para a linha 1060 imprimir a letra correspondente. C e L são respectivamente a linha e a coluna onde a mensagem é impressa. O laço da linha 1010 faz com que todas as letras sejam lidas, uma a uma, até o final de **A$**. **N** é a posição absoluta do "cursor" na tela.

N é usado em lugar de L e C para evitar que a mensagem ultrapasse os limites da tela, "pulando" de linha. Observe que é **N** que é incrementado na linha 1070 e que **L** e **C** têm seus novos valores calculados a partir de **N**, na linha 1040.

As últimas linhas do programa fazem uma pequena moldura colorida para a mensagem, lembrando ao usuário que se trata da tela gráfica.

As letras aparecerão com cores aleatórias — geralmente tons de amarelo. Isto é inevitável devido à maneira com que o Apple codifica as cores.

Para ver e editar os novos caracteres usando o editor, apague o programa com **NEW** e carregue o gerador. Antes de executá-lo, acrescente as próximas linhas, que dão a ele uma nova função, disponível através de "M".

```
195  IF K$ = "M" THEN  GOSUB 21
10: GOTO 50
```

T

O TRS-Color tem dois comandos — **GET** e **PUT** —, que permitem a criação e o controle de blocos gráficos. Como esse micro guarda os valores correspondentes ao padrão dos blocos dentro de matrizes, o único limite ao número de blocos é o tamanho da memória. Isso significa que podemos criar tantos blocos quantos couberem em 32k.

Ao contrário dos demais computadores, o TRS-Color não permite que mudemos o formato de seus caracteres da ROM — ou seja, os caracteres disponíveis por intermédio do teclado. Portanto, não podemos alterar os tipos com que são escritas as listagens e as mensagens de erro. É possível, porém, produzir blocos gráficos que sejam ao mesmo tempo letras e dar a estas o formato que se quiser.

Um programa para isso deveria ter uma série de comandos **IF INKEY$ = "X" THEN...**, um para cada letra, o que o tornaria muito lento. Se você optar por uma operação dessas para alguns caracteres, empregue o comando **PMODE1** e as linhas **DATA** do programa do Spectrum.

UM COMPACTADOR DE PROGRAMAS

Ao se digitar um programa em BASIC, é conveniente colocar espaços em branco entre os comandos, as variáveis e os dados, para tornar mais fácil a leitura das linhas.

Necessárias para nossa própria orientação, as linhas **REM** são especialmente importantes quando estamos depurando o programa. O problema é que os espaços entre as palavras e as linhas **REM** diminuem a velocidade de execução do programa e ocupam um grande espaço na memória.

Quando o programa está funcionando bem e desejamos obter um máximo de rendimento, temos que apagar todas essas linhas e espaços. Isso, porém, exige muito esforço e, o que é pior, pode introduzir novos erros no programa.

O programa em código de máquina que apresentamos aqui faz todo esse trabalho para você. Uma vez que seu programa esteja funcionando, execute a rotina em código para compactá-lo.

T

A seguir, apresentamos um compactador de programas para o TRS-Color. Caso você use o monitor, e não o Assembler, o endereço inicial é 30000.

```
10   ORG 30000
20   LDU 25
30   LDD 27
40   PSHS D
50   BRA LONE
60   LTWO LDA ,X+
70   BNE LTHR
80   LDU ,U
90   LONE CLR ICOM,PCR
100   LDD ,U
110   BNE LFOU
120   LDX 27
130   STX 29
140   STX 31
150   PULS D
160   SUBD 27
170   JSR $B4F4
180   RTS
190   LFOU LEAX 4,U
200   BRA LTWO
210   LTHR CMPA #32
220   BNE LFIV
230   TST ICOM,PCR
240   BNE LTWO
250   LDD #1
260   PSHS D,X,U
270   BSR SHIFT
280   PULS D,X,U
290   LEAX -1,X
300   BRA LTWO
310   LFIV CMPA #34
320   BNE LTWE
330   LDB ICOM,PCR
340   EORB #1
350   [illegible] ICOM,PCR
360   BRA LTWO
370   LTWE CMPA #134
380   BNE LSIX
390   LDA -2,X
400   CMPA #255
410   BEQ LTWO
420   LDB ICOM,PCR
430   EORB #2
440   STB ICOM,PCR
450   BRA LTWO
460   LSIX CMPA #130
470   BEQ LSEV
480   CMPA #131
490   BNE LTWO
500   LSEV LDA -2,X
510   CMPA #255
520   BEQ LTWO
530   LDD ,U
540   LEAX -1,X
550   PSHS X,U
560   SUBD ,S++
570   TFR X,Y
580   LDX ,U
590   LEAX -1,X
600   PSHS D,X
610   TFR Y,D
620   SUBD 4,S
630   CMPB #4
640   BNE LNIN
650   PULS D
660   ADDD #4
670   PSHS D
680   LNIN BSR SHIFT
690   PULS D,X,U
700   LBRA LONE
710   ICOM FCB 0
720   SHIFT LDD ,U
730   BEQ SHTWO
740   LDX ,U
750   SUBD 2,S
760   STD ,U
770   TFR X,U
780   BRA SHIFT
790   SHTWO LDD 4,S
800   SUBD 2,S
810   TFR D,U
820   LDX 4,S
830   SHTHR LDA ,X+
840   STA ,U+
850   CMPX 27
860   BLO SHTHR
870   LDD 27
880   SUBD 2,S
890   STD 27
900   RTS
910   END
```

COMO FUNCIONA

A primeira instrução **LDU 25** carrega o registro U com o conteúdo das posições 25 e 26 da memória. Estas contêm o valor de uma variável do sistema que corresponde ao endereço inicial do programa em BASIC. **LDD 27** carrega o registro D com o conteúdo das posi-

Apague as linhas REM e os espaços inúteis, tornando seus programas em BASIC mais rápidos e econômicos. Existe um programa em código para isso.

- LOCALIZE OS COMANDOS REM E OS ESPAÇOS
- COMO USAR SINALIZADORES
- COMO COMPRIMIR UM PROGRAMA NA MEMÓRIA

ções 27 ■ 28, que contêm igualmente o valor de uma variável do sistema. Desta vez, porém, trata-se de um apontador, que indica o endereço da primeira posição livre da memória, após ■ fim do programa em BASIC. Seu valor é guardado na pilha por **PSHS D**.

Mais adiante, durante a execução do programa, será calculado ■ número de bytes economizado pelo processo de compressão. Assim, é preciso que ■ computador saiba onde o antigo programa em BASIC termina. A instrução **BRA** (**BR***anch* **A***lways*, ou "desvie sempre") transfere a execução para o meio da sub-rotina seguinte, no rótulo **LONE**.

COMO USAR SINALIZADORES

A linha **CLR ICOM, PCR** limpa a área de armazenamento formada pela instrução **FCB0** que se segue ao rótulo **ICOM**. O comando **PCR** (*Programa Counter Relative*) faz com que o endereçamento utilizado possibilite a realocação do programa quando necessário.

FCB (*Form Constant Byte*) reserva um byte para ■ armazenagem de dados. Qualquer tipo de dado pode ser guardado ali, mas, no nosso caso, armazenaremos um par de indicadores que o programa usará para saber se está manipulando uma variável alfanumérica. Quando se tratar de um cordão, os espaços em branco não devem ser apagados. Não é conveniente também que as palavras que ■ BASIC escreve na tela apareçam juntas. Os bits da posição de memória rotulada como **ICOM** serão usados como sinalizadores, do mesmo modo que é feito no registro indicador de condições. Os indicadores serão estabelecidos (ou ligados) quando o programa encontrar uma variável alfanumérica, voltando ■ valer zero quando o programa estiver cuidando de outras expressões. Desse modo, ao fazer ■ compactação, podemos verificar os bits de **ICOM** para saber como proceder.

Existe uma instrução parecida com **FCB**. É **FDB** (*Form Double Byte*) que reserva dois bytes para colocar dados.

O 0 após **FCB** coloca inicialmente 00 nesse byte (se o 0 for colocado após **FDB**, ■■ dois bytes reservados receberão 00 00). **CLR ICOM, PCR** limpa ■ conteúdo do byte a cada linha.

COMO ATUALIZAR ■ APONTADORES

LDD,U coloca no registro D o conteúdo do endereço contido em U. Em outras palavras, ele posiciona os dois primeiros bytes do programa em D.

Os dois primeiros bytes de um programa em BASIC correspondem ■■ endereço do início da linha seguinte. Quando esses dois bytes forem 00 00, chegamos ao final do programa. **BNE** (desvio, se diferente de zero) verifica essa condição. Enquanto não ■ tratar do final do programa, esses dois bytes serão diferentes de 00 00, ■ ■ desvio não será realizado. **LDX** coloca o conteúdo das posições de memória 27 e 28 no registro X. Essas posições contêm ■ endereço do início da "área das variáveis", isto é, o primeiro byte livre depois do programa BASIC.

Esse apontador será atualizado repe-

tidas vezes à medida que ■ programa for comprimido. STX ■ ■ STX 31 colocam esse endereço nas posições 29 e 30, e 31 e 32. Esses dois apontadores são variáveis do sistema que correspondem ao endereço inicial da tabela de apontadores das matrizes e ao endereço final da "área" do BASIC. As variáveis serão definidas ■ as matrizes montadas somente quando o BASIC for executado. Assim, aquelas duas variáveis do sistema devem ficar apontando para ■ endereço do primeiro byte livre após o final do programa em BASIC, enquanto este é compactado.

ESCREVA NA TELA

O valor contido no apontador do final da área BASIC (colocado na pilha quando ■ programa ainda tinha o tamanho original, no início da rotina em código) é recuperado da pilha. Em seguida, ■ valor que fica no apontador após a compressão do programa será subtraído daquele que acaba de voltar da pilha ■ o resultado, armazenado em D.

JSR $B4F4 transfere ■ programa para uma sub-rotina da ROM que toma ■ valor do registro D, converte-o em decimal ■ o imprime na tela. Como podemos observar, o registro D contém o número de bytes economizados pela compressão do programa. Quando ■ processo terminar, ■ economia de espaço será impressa ■ tela. **RTS** retorna ■ BASIC.

COMO PERCORRER A MEMÓRIA

Se o programa compactador não chegar ao final do programa em BASIC, **BNE LFOU** enviará este último para a próxima ocorrência do rótulo **LFOU**. O comando **LEA** significa "carregar o endereço efetivo" (Load Efective Adress). Neste caso, ele opera sobre o registro X. **LEA X 4,U** toma o endereço em U, soma 4 ■ coloca o resultado em X. Os dois primeiros bytes de qualquer linha BASIC são o endereço inicial da linha seguinte; os dois bytes imediatamente posteriores contêm o número da linha. Deste modo, ■ instrução move ■ microprocessador para o início dos códigos do BASIC. Em seguida, o programa vai para o rótulo **LTWO**.

LDA,X + coloca o primeiro byte do BASIC no acumulador. O sinal "+" significa que ■ registro X é incrementado após ■ execução da instrução. Isso faz com que o valor contido em X corresponda ao byte seguinte do programa.

BNE LTHR desviará para o rótulo **LTHR** se ■ conteúdo do acumulador não for zero. Um byte igual ■ 0 (zero) corresponde ao final da linha BASIC. Quando o programa chegar ao fim da linha, o desvio não ocorrerá ■ o conteúdo da memória cujo endereço está em U será colocado no registro U por **LDU, U**. No início, o conteúdo da variável do sistema que aponta para ■ começo do BASIC foi colocado em U. No TRS-Color, os dois primeiros bytes de uma linha correspondem ao endereço da linha seguinte. Cada vez que ■ programa em código chega ao fim de uma linha BASIC, o registro U é atualizado de modo a corresponder ao endereço inicial da próxima linha. A seguir, é realizado ■ comando de limpeza **CLR**.

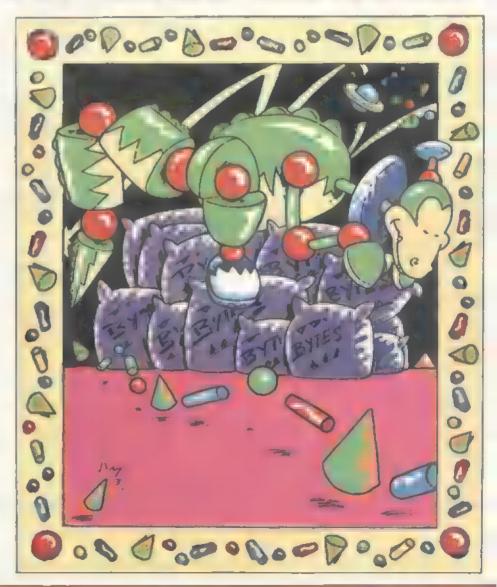
CUIDE DOS ■

Se ainda não ■ trata do final de uma linha BASIC, ocorre desvio e o comando **CMPA #32** verifica se ■ byte colocado no acumulador corresponde a um espaço. O código ASCII do caractere de espaço é 32. Caso se trate de um espaço, o valor do acumulador será igualado a 0 (zero) pelo comando **CMPA #32**, de forma que ■ instrução de desvio **BNE LF,V** não será executada.

TST ICOM,PCR verifica o estado da posição de memória rotulada como **ICOM**. Se qualquer um dos bits indicadores em **ICOM** valer 1, é sinal de que se trata de um cordão, de modo que o espaço não deve ser eliminado e **BNE LTWO** voltará ■ início do programa. Caso o espaço esteja entre dois comandos BASIC, o processador passará para ■ próxima instrução: **LDD #1**.

■ USAR A PILHA

LDD #1 coloca 1 no registro D. **PSHSD,X;U** transporta o conteúdo des-

ses registros para a pilha da máquina. Pode parecer desnecessário pôr o registro D na pilha, já que sabemos que ele contém 1; contudo, ■ instrução seguinte, **BSR SHIFT**, desvia para a sub-rotina **SHIFT**. Esta desloca o programa aproveitando o espaço deixado pelos caracteres apagados. Ela é usada quando apagamos espaços — apenas um byte — ■ quando apagamos linhas **REM** — vários bytes. O valor do registro D corresponde ao número de bytes apagados e é usado pela sub-rotina. No caso de uma linha **REM**, esse valor pode chegar a 255 bytes.

Usamos um comando de salto relativo, **BSR**, em vez de um de salto absoluto, **JSR**, de maneira que o programa seja realocável na memória. O comando **JSR** é mais adequado quando usamos sub-rotinas de ROM, que são fixas.

Depois que ■ rotina na **SHIFT** for executada, os valores de D,X e U serão recuperados por **PULS D,X,U**. Note que D, X ■ U são especificados na mesma ordem em que estão guardados na pilha.

Não importa a ordem em que definimos os registros. O microprocessador tem uma ordem pré-determinada para guardar e recuperar grupos de registros. A ordem de armazenamento é: byte menos significativo do registro PC; byte mais significativo de PC; bytes de S ou U, na mesma ordem baixo-alto; Y, X, DP, A, B e CC. A ordem de recuperação é ■ inversa: CC primeiro, PC por último.

Os registros U ■ S são os apontado■■ das duas pilhas — U para a pilha do usuário e S para a pilha da máquina. Obviamente, não podemos colocar o valor de um apontador dentro de sua própria pilha, nem podemos recuperar um valor da pilha, colocando-o dentro de seu apontador. Este é o motivo de termos colocado U ou S na ordem anterior. A própria instrução informa de que pilha se trata. **PSHS** ■■■ a pilha de máquina e **PSHU** e **PULU**, a pilha do usuário.

LEAX -1,X decrementa o registro X, subtraindo 1 de ■■■ conteúdo. Isso é feito porque ■ programa acaba de ser deslocado um byte para baixo na memória. X é o apontador da posição em que nos encontramos dentro da listagem BASIC. Ele apontava para o espaço que foi apagado, mas o byte seguinte da linha acaba de ser transferido para o mesmo local. Para considerar esse byte, temos que examinar a posição de memória novamente, em vez de observarmos ■ byte seguinte. Quando **BRA LTWO** volta ao princípio novamente, a primeira instrução incrementa ■ registro X, mudando para o próximo byte.

COMO PROCURAR CORDÕES

Se o programa não encontrar um caractere de espaço na instrução **CMPA #32**, a instrução **BNE LFIV** desviará para uma pequena rotina que procura aspas. **CMPA #34** compara o conteúdo do acumulador com ■ código ASCII das aspas. Se um caractere de aspas for encontrado, **CMPA #34** resultará 0; assim, a condição necessária à execução de **BNE LTWE** não será satisfeita ■ **LDB ICOM, PCR** carregará o registro ■ com ■ conteúdo da posição de memória onde os sinalizadores foram guardados.

EORB #1 executa ■ operação lógica "ou exclusivo" entre B e 1. Essa operação existe também em BASIC. **STB ICOM, PCR** coloca ■ resultado de volta no byte de sinalizadores; o bit zero é ativado como sinal de que um caractere de aspas foi encontrado. Se esse bit **ICOM** já era igual a 1, **EORB #1** retornará o valor 0: ■■ o bit era 0, ■ resultado será 1. Assim, se o bit zero estava ligado, essa operação o desligará. Caso contrário, ela o ligará.

A FUNÇÃO ■■■ ■■■■■

Se observarmos agora como as aspas envolvem as mensagens dentro de um programa, veremos que um cordão começa depois de um número ímpar de aspas ■ termina após um número par. Alternando o bit zero do byte de sinalizadores toda vez que um caractere de aspas for encontrado, o comando **EOR** deixará o bit valendo 1 após um número ímpar de aspas e 0 após um número par. O sinalizador ligado indica, portanto, que o programa ■■ encontra em um cordão; desligado, ele revela que não se trata de um cordão. Esta é exatamente a condição que é testada por **TST ICOM, PCT** seguido de **BNE LTWO**.

Uma vez alterada a condição do sinalizador de aspas, **BRA LTWO** retorna ao princípio novamente, passando ao próximo byte da listagem BASIC.

Se o byte não estivesse entre aspas, a condição necessária de **BNE LTWE** teria sido satisfeita, e o programa prosseguiria com novas verificações.

■■■■■■■DO LINHAS DATA

As linhas **DATA** podem conter cordões; de forma que é melhor não comprimi-las também.

A instrução **CMPA #134** verifica se o byte no acumulador é o código do comando **DATA**. Caso seja, a condição necessária de **BNE LSIX** não será satisfeita.

Contudo, há outra situação em que o código 134 pode aparecer. A função **LOG** tem o código FF 86, em hexadecimal, ou 255 seguido de 134, em decimal. Assim, é preciso verificar se o byte anterior não é 255.

Como você deve estar lembrado, **LDA ,X+** incrementa o registro X após colocar nele o valor do acumulador. Desse modo, o registro X aponta agora para o byte posterior ao que está sendo considerado. Para verificar o byte anterior, ■ acumulador deve ser carregado com o conteúdo da memória cujo endereço é o resultado de X menos 2. A instrução **LDA -2,X** faz isso.

O comando **CMPA #255** verifica então se o byte anterior é 255. Se FF for encontrado, **BEQ LTWO** retornará ao princípio.

Se FF não for encontrado, isso significa que $86 é o código de um comando **DATA** ■ que, portanto, um sinalizador em **ICOM** deve ser ativado. Obviamente, não se trata do bit zero, que conta as aspas.

Assim, ■ instrução **LDB ICOM, PCR** coloca o valor de **ICOM** em B. Por sua vez, **EORB #2** realiza um "ou exclusivo" (**XOR**) entre B e 2. Isso ativa o bit dois.

COMO PROCURAR LINHAS REM

CMPA #130 verifica se o próximo byte é o código de uma instrução **REM**. Se for, **BEQ LSEV** levará o microprocessador ao rótulo **LSEV**. Se não for, **CMPA #131** verificará se o byte é um apóstrofo que funciona como **REM**.

Se, ao contrário, nenhum desses códigos for encontrado, a instrução **BNE LTWO** voltará ao princípio, passando ao próximo byte.

Caso um desses sinais seja encontrado, o programa deve verificar se não se trata de outras funções, cujos códigos são precedidos de FF. Isto é feito pelas três instruções seguintes: **LDA -2,X**, **CMPA #255** ■ **BEQ LTWO**.

Depois disso, o registro D é carregado com o valor de U, que corresponde ao endereço da próxima linha BASIC.

Por seu turno, ■ instrução **LEAX -1,X** decrementa o registro X, fazendo com que o apontador indique a posição do comando ■■■■ ou do apóstofro (nosso objetivo é apagar o trecho que vai desse ponto até ■ fim da linha). A seguir, **PSHS** guarda X ■ U ■■ pilha e **SUBD ,S ++** subtrai o valor dos dois últimos bytes da pilha do registro D; o resultado é colocado em D mesmo. A

instrução também incrementa em duas vezes o apontador da pilha.

Como você já deve saber, o conteúdo de U é guardado antes de X na pilha. **SUBD ,S++** subtrai do valor de D a última coisa que foi guardada na pilha da máquina.

O sinal "++" após S incrementa o registro S (apontador da pilha da máquina) em duas vezes, ou seja, soma 2 a ele. Isso retira o valor de X — dois bytes — da pilha.

TFR X,Y transfere (copia) o conteúdo de X para Y. O registro Y não é usado para mais nada nesse programa, servindo apenas como um registro temporário para guardar o valor de X.

LDX ,U coloca o valor de U em X. U contém o endereço inicial da próxima linha. **LEAX -1,X** decrementa esse valor, de modo a fazê-lo corresponder ao endereço final da linha atual. **PSHS D,X** guarda o conteúdo desses registros na pilha para que possam ser usados pela rotina **SHIFT**.

TFR Y,D transfere para D ■ valor de Y. Assim, a posição do código da instrução **REM** está agora em D. **SUBD 4,S** subtrai de D o valor correspondente aos dois bytes que se encontram a partir do quinto byte da pilha. (S aponta para o primeiro byte da pilha. A expressão 4,S soma ■ a S, de forma que este passa ■ apontar para o quinto byte). Observe ■ que foi feito antes: esse valor é o antigo conteúdo do registro U, endereço inicial da próxima linha do programa BASIC.

Isso significa que o endereço inicial da linha é subtraído do endereço do código da instrução **REM**. O resultado nos dá o número de bytes entre o início da linha e a instrução. O que precisamos saber é se a instrução **REM** está no começo ou no meio da linha. No primeiro caso, podemos apagar também ■ número da linha. Isso é verificado por **CMPB #4**.

Se a instrução **REM** não estiver no começo da linha, **BNE LNIN** irá diretamente à instrução que chama a sub-rotina **SHIFT**. Contudo, se a **REM** estiver no início da linha, o resultado da operação será 4 ■ o último valor colocado na pilha será recuperado no registro D por **PULS D**. Soma-se 4 a D para que os quatro bytes iniciais da linha ■■■ — correspondentes ao endereço inicial da linha seguinte e ao número da linha — também sejam apagados no processo de compressão. Finalmente, o conteúdo de D é devolvido à pilha.

A sub-rotina **SHIFT** é chamada por **BSRSHIFT** e o conteúdo original dos registros é recuperado da pilha. ■■■A **LONE** volta ao início do programa.

Um desvio longo — **L**ong **BRA**nch — é usado porque o salto é maior que 128 bytes.

A ROTINA DE COMPRESSÃO

O comando **LDD ,U** coloca em D o conteúdo do endereço apontado por U — em outras palavras, D vai conter o endereço inicial da próxima linha do programa em BASIC. Se esse valor for zero, teremos chegado ao fim do programa e a instrução **BEQ LTWO** sairá do laço principal do programa.

Se não se tratar do fim do programa em BASIC, ■ mesmo valor será colocado em X por **LDX ,U**. O valor correspondente aos dois bytes localizados a partir do terceiro byte da pilha será então subtraído por **SUBD 2,S** do endereço em D (endereço inicial da próxima linha).

Por outro lado, devemos ter em mente que estamos no meio de uma sub-rotina e os dois bytes correspondentes ■■ endereço de retorno da sub-rotina foram colocados ■■ pilha da máquina — daí a necessidade do **2,S**.

O resultado dessa operação é o endereço inicial que a próxima linha BASIC ocupará, uma vez feita ■ compressão. Ela é colocada na posição de memória apontada por U, que corresponde aos dois primeiros bytes da linha BASIC. Estes, por sua vez, indicam ■ endereço da próxima linha. Em outras palavras, estamos fazendo com que esse valor coincida com as mudanças de endereço causadas pela compressão.

O valor de X que corresponde ao endereço inicial antigo é colocado em U e **BRA SHIFT** volta novamente ao início da sub-rotina **SHIFT**.

A única saída desse laço é ■ **BEQ**, que só será executado quando todo ■ programa BASIC tiver passado por esse tipo de modificação. Quando ■ preparação for finalmente completada, podemos comprimir nosso programa. O comando **LDD 4,S** transporta para D o valor de X guardado ■■ pilha, isto é, ele coloca o endereço final da linha atual se estivermos eliminando um comentário **REM**, ou ■ posição atual ■■ linha, caso estejamos eliminando um espaço.

SUBD 2,S subtrai o número de bytes que serão eliminados.

O resultado é a posição que o próximo byte da linha deve ocupar; ele é colocado em U pela instrução **TFR D,U**.

LDX 4,S recupera outra vez o antigo valor de X guardado ■■ pilha. Assim, ■ endereço antigo do próximo byte está em X e ■ novo em U.

LDA ,X+ coloca o byte apontado por X em A e incrementa o valor de X. **STA ,U+** põe o mesmo valor na posição de memória apontada por U e incrementa o valor de U. Assim, o próxi■■ byte do programa não comprimido é transferido para sua nova posição, no programa comprimido. Além disso, os dois apontadores X e U são incrementados para que apontem para o próximo byte e para a próxima posição.

CMPX #27 compara ■ conteúdo de X com o conteúdo da variável do sistema que fica nos endereços 27 e 28. Ela corresponde ao endereço inicial da área das variáveis, que deve ficar logo acima do final do programa em BASIC. Essa instrução verifica se chegamos ao final da listagem. Se isto ainda não aconteceu, **BLO** — desvio se for menor — desviará para o rótulo **SHTHR**, continuando ■ compressão.

Quando todo o programa tiver sido comprimido, o microprocessador executará **LDD 27**. Essa instrução coloca o apontador do final do programa ■■ registro D. **SUB 2,S** subtrai o número de bytes eliminados e **STD 27** coloca o apontador atualizado de volta em seu lugar. **RTS** retorna à rotina principal.

COMO USAR O COMPRESSOR

Esse programa pode ser montado com qualquer endereço inicial, desde que o local tenha sido protegido. Depois de gravar ■ programa-fonte ■ montar o compressor, podemos usar **NEW** para eliminar o Assembler e então carregar o programa que vamos compactar. Para executar ■ compressão, use:

```
DEF USR0 = endereço inicial
```

onde ■ endereço inicial é ■ origem. Depois, digite:

```
PRINT USR0(0)
```

Devemos, contudo, ser muito cuidadosos. Não será possível editar o programa depois da compressão; portanto, ele deve ter sido exaustivamente testado. Muita atenção com os comandos **GOTO** e **GOSUB** que ■■ referiam a linhas **REM**: eles devem ser mudados.

Para gravar o compressor em código, digite:

```
CSAVEM "CMPRSS" endereço
inicial, endereço final,
endereço inicial
```

Para carregar ■ programa de volta, use:

```
CLOADM "CMPRSS"
```

CONJUNTOS DE BLOCOS GRÁFICOS (2)

- QUANDO USAR BLOCOS GRÁFICOS
- COMO COMBINAR BLOCOS PARA FORMAR FIGURAS
- CRIAÇÃO DE UM CENÁRIO
- BANCOS DE BLOCOS

Com um conjunto de blocos gráficos, será fácil criar as mais diversas figuras. Você precisará apenas colocar os blocos na posição correta e acrescentar alguns detalhes no fundo.

Já falamos bastante sobre a criação, proteção, edição e gravação de conjuntos de blocos definidos pelo usuário. Mostramos também ao usuário do Spectrum como exceder o limite de 21 blocos inicialmente imposto. Neste artigo, o segundo de uma série de três, avançamos as explicações sobre o uso de blocos gráficos na criação de figuras na tela de seu microcomputador.

POR QUE USAR BLOCOS GRÁFICOS

Suponhamos que você queira criar uma tela representando uma floresta — para servir de cenário a um jogo de ação, por exemplo. Existem basicamente dois caminhos a seguir: usar os comandos gráficos do BASIC para desenhar cada parte da figura ou, então, combinar blocos gráficos.

Se você escolher a primeira alternativa, terá um trabalho enorme para criar a mata que faz parte do cenário, pois precisará desenhar cada tronco e cada copa de árvore. Além disso, as árvores ficarão muito parecidas umas com as outras.

Se você criar um bloco gráfico, ou vários deles que, combinados, formem uma árvore, poderá colocá-los na tela, repetidas vezes, nas posições que quiser. Você evitará, desse modo, o trabalho de

especificar cada detalhe ao acrescentar mais uma árvore na tela.

Existem ainda outras vantagens no uso de UDG. Uma delas é ■ economia de tempo. Os desenhos compostos por blocos gráficos, não importa se mais ou menos complicados, são traçados na tela numa velocidade maior que os produzidos com comandos gráficos do BASIC. Assim, ■ compusermos uma figura com blocos gráficos, teremos que esperar bem menos tempo por sua aparição na tela.

LIBERDADE DE ESCOLHA

Outra vantagem é ■ facilidade de variar o tamanho e ■ forma da figura. No desenho da floresta, por exemplo, poderíamos mudar o tamanho de uma árvore aumentando ou diminuindo o número de "blocos de tronco" e, ainda, alterar sua copa usando uma combinação diferente de "blocos de folhagem". Os comandos gráficos do BASIC podem fazer a mesma coisa, mas, com os blocos, a tarefa é bem mais fácil.

Uma vez criados, os blocos gráficos permanecem na memória até serem modificados — ou o micro ser desligado. No caso do Spectrum, se dispusermos de vários bancos, precisaremos "ligar" o banco que estiver sendo usado naquele momento. Mas isto é só uma questão de modificar o apontador de UDG.

Poderemos, assim, usar os blocos que criamos quantas vezes quisermos dentro de um programa: o único limite é a memória. Voltando ao nosso exemplo, será tão fácil criar uma floresta quanto uma única árvore, utilizando UDG.

É claro que o emprego de blocos gráficos também apresenta desvantagens. Para começar, temos que definir todos os blocos, recorrendo ao gerador ou digitando várias linhas **DATA**. Vale lembrar, porém, que, usando comandos gráficos, teremos ainda mais trabalho.

Outra desvantagem é a ocupação de parte da preciosa memória do micro para guardar ■ blocos e, pior, a menos que sejamos cuidadosos, acabamos fazendo-o duas vezes. No próximo artigo, veremos como contornar o problema.

QUANDO VALE A PENA USAR UDG

Por todas estas razões, é mais vantajoso usar blocos gráficos em certas figuras do que em outras.

Como regra geral, se ■ cenário inclui uma série de objetos semelhantes, desenhados várias vezes, ou se algum tipo de figura aparece repetidamente no decorrer do programa, o uso de UDG vai nos poupar tempo e trabalho.

Se, por outro lado, queremos um desenho muito detalhado, que usaremos apenas uma vez durante o programa, pode ser melhor ficarmos com os comandos gráficos do BASIC.

Uma parede de tijolos, por exemplo, poderá ser desenhada com muito mais facilidade se utilizarmos um ou vários blocos repetidas vezes. Uma alternativa seria traçar diversas linhas sobre uma região colorida para imitar os espaços entre os tijolos.

No nosso cenário de floresta, já vimos que podemos fazer uma boa economia de tempo criando UDG para as árvores que nele aparecem. Os animais também podem ser desenhados com blocos gráficos, principalmente se formos utilizá-los novamente no decorrer do programa, ou se quisermos animá-los ou desenhá-los várias vezes.

UMA [illegible] NA TELA

Apresentaremos a seguir alguns programas que criam os caracteres para desenhar o cenário já mencionado.

As linhas **DATA** para ■ Spectrum e ■ MSX são ■ mesmas. Elas começam na linha 1300 e estão listadas logo após ■ programa do MSX.

```
 10 CLEAR 63500
110 POKE 23676,255
120 FOR n=USR "a" TO USR "r"+7
: READ a: POKE n,a: NEXT n
260 POKE 23676,249
270 FOR n=USR "a" TO USR "■"+7
: READ a: POKE n,a: NEXT n
290 POKE 23676,248
300 FOR n=USR "a" TO USR "o"+7
: READ a: POKE n,a: NEXT n
410 BORDER 1: PAPER 8: CLS
420 FOR n=1 TO 8: PRINT  PAPER
5;" ";TAB 31;" ": NEXT ■
430 FOR n=1 TO 14: PRINT
PAPER 4;" ";TAB 31;" ": NEXT n
440 PLOT 0,110: DRAW 142,-100,
-PI/3: PLOT 160,110: DRAW -60,
```

```
-42;PI/3
460 POKE 23676,255: INK 2
470 PRINT AT 19,20;: FOR n=144
TO 155: PRINT CHR$ n;: NEXT n
480 PRINT AT 20,20;CHR$ 156;
CHR$ 157;CHR$ 158;CHR$ 159;
CHR$ 159;CHR$ 159;CHR$ 159;
CHR$ 159;CHR$ 160;CHR$ 159;
CHR$ 159;CHR$ 159
490 FOR n=0 TO 31: PRINT  INK
1; PAPER 4;CHR$ 161;: NEXT n
760 INK 7
770 POKE 23676,249
780 PRINT AT 8,9;CHR$ 144;CHR$
145;CHR$ 146;CHR$ 147;AT 9,9;
CHR$ 148;CHR$ 149;CHR$ 150;
CHR$ 151;CHR$ 152;AT 10,10;
CHR$ 153;CHR$ 154;CHR$ 155;
CHR$ 156
800 POKE 23676,248
810 LET x=6: LET y=14: GOSUB
840: LET x=5: LET y=18: GOSUB
840
820 LET x=4: LET y=0: GOSUB
845: LET x=4: LET y=3: GOSUB
845
830 GOTO 850
840 PRINT  INK 4;AT x,y;CHR$
151;CHR$ 152;CHR$ 153;CHR$ 154
; INK 2;AT x+1,y+1;CHR$ 155;
CHR$ 156;AT x+2,y+1;CHR$ 157;
AT x+3,y+1;CHR$ 157;AT x+4,y+1
;CHR$ 157;AT x+5,y+1;CHR$ 158:
 RETURN
 845 PRINT  INK 4;AT x,y;CHR$
 144;CHR$ 145;AT x+1,y;CHR$ 146
 ;CHR$ 147;AT x+2,y;CHR$ 148;
 CHR$ 149; INK 2;AT x+3,y+1;
 CHR$ 150;AT x+4,y+1;CHR$ 150:
 AT x+5,y+1;CHR$ 150: RETURN
 970 INK 0
1300 REM CROCODILO
1310 DATA 0.0,1,7,15,15,9,5,0,0
,128,192,248,255,127,95,1,3,6,1
2
1320 DATA 62,255,255,255,192,22
4,176,159,191,255,255,255
1330 DATA 0,0,0,0,0,248,252,255
,0,0,0,0,0,0,1,207,0,0,0,1,15,1
27,255,255
1340 DATA 0,3,63,255,255,255,25
5,255,127,255,255,255,255,254,2
49,247,248
1350 DATA 255,255,255,255,15,25
5,255
1360 DATA 0,224,254,255,255,255
,255,255,0,0,0,192,248,255,255,
255,0,2,4,7,7
1370 DATA 3,0,0,21,1,164,73,255
,255,0,0
1380 DATA 255,127,63,63,255,255
,127,31,255,255,255,255,255,255
,255,255
1390 DATA 239,239,239,239,239,2
47,247,247,60,255,255,255,255,2
55,255,255
1780 REM ELEFANTE
1790 DATA 0,0,0,8,28,25,51,51,0
,0,0,126,255,193,253,253,0,0,0,
0,0,255,255
1800 DATA 255,0,0,0,0,0,248,252
,254
1810 DATA 102,111,111,111,125,5
7,26,0,253,251,251,251,231,31,1
5,15,255,255
1820 DATA 255,255,255,255,255,2
55,254,255,255,255,255,255,255,
255
1830 DATA 0,0,128,64,32,16,12,0
,15,15,15,14,14,14,14,31,255,24
0,224,224
1840 DATA 224,224,224,224,255,6
3,59,59,57,57,121,248
1850 DATA 0,0,128,192,224,224,1
28,0
1860 REM ARVORE 1
```

```
1870 DATA 0,0,0,0,1,1,3,7,0,0,0
,0,224,240,248,248,15,63,63,63,
31,15,3,3
1880 DATA 252,254,254,254,254,2
54,252,252
1890 DATA 3,3,3,3,3,1,0,0,248,2
48,248,248,248,248,240,96,96,96
,96,96,96,96
1900 DATA 96,96
1910 [illegible] ARVORE 2
1920 DATA 0,3,15,31,127,127,63,
1,7,15,255,255,255,255,255,255,
15,63,255,255
1930 DATA 255,255,255,255,0,128
,248,248,248,248,240,224
1940 DATA 255,227,96,48,24,25,1
3,15,252,240,96,96,192,192,128,
128,7,7,7,7,7
1950 DATA 7,7,7,7,7,7,15,15,15,
31,63
```

```
5 CLEAR 200,&HC999
10 [illegible] I=0 TO 367
20 READ A:POKE &HD100+I,A
30 NEXT I:COLOR 15,4,4
110 SCREEN 2
120 CIRCLE (255,191),160,2,0,6.
28,.6:PAINT(230,191),2
130 CIRCLE(0,191),143,2,0,6.28,
.4:PAINT(10,191),2
140 CIRCLE(0,191),143,12,0,6.28
,.4
200 FOR I=1 TO 51
210 READ A,B,C:FOR J=0 TO 7
220 VPOKE [illegible](12)+A*8+J,PEEK(&
HD100+B*8+J)
230 VPOKE BASE(11)+A*8+J,C
```

```
240 NEXT J,I
250 FOR K=1 TO 6:READ D(K):NEXT
 K
[illegible] FOR I=1 TO 8:READ A(I),B(I)
,C(I):NEXT I:FOR K=1 TO 6:FOR I
=1 TO 8:FOR J=0 TO 7
270 VPOKE BASE(12)+(A(I)+D(K))*
8+J,PEEK(&HD100+B(I)*8+J)
280 VPOKE BASE(11)+(A(I)+D(K))*
8+J,C(I)
290 NEXT J,I,K
300 [illegible] K=1 TO 8:READ D(K):NEXT
 K
310 [illegible] I=1 TO 9:READ A(I),B(I)
,C(I):NEXT I:FOR K=1 TO 8:FOR I
=1 TO 9:FOR J=0 TO 7
320 VPOKE BASE(12)+(A(I)+D(K))*
8+J,PEEK(&HD100+B(I)*8+J)
330 VPOKE BASE(11)+(A(I)+D(K))*
8+J,C(I)
340 NEXT J,I,K
400 GOTO 400
3000 DATA 692,0,98,693,1,98,694
,2,98,695,3,98
3010 DATA 696,4,98,697,5,98,698
,6,98,699,7,98
```

```
3020 DATA 700,8,98,701,9,98,702
,10,98,703,11,98
3030 DATA 724,12,98,725,13,98,7
26,14,98,727,15,98
3040 DATA 728,15,98,729,15,98,7
30,15,98,731,15,98
3050 DATA 732,16,98,733,15,98,7
34,15,98,735,15,98
3060 DATA 754,17,78,755,17,78,7
56,17,78,757,17,78,758,17,78,75
9,17,78
3070 DATA 760,17,78,761,17,78,7
62,17,78,763,17,78
3080 DATA 764,17,78,765,17,78,7
66,17,78,767,17,78
3090 DATA 500,18,226,501,19,226
,502,20,226,503,21,226
3100 DATA 532,22,226,533,23,226
,534,24,226,535,25,226
3110 DATA 536,26,226,565,27,226
,566,28,226,567,29,226,568,30,2
26
3115 DATA -32,-4,36,75,80,147
3120 DATA 342,31,36,343,32,36,3
74,33,36,375,34,36
3130 DATA 406,35,36,407,36,36,4
39,37,96,471,37,96
3135 DATA -44,-20,-15,60,84,96,
157,163
3140 DATA 327,38,36,328,39,36,3
29,40,36,330,41,36
3150 DATA 360,42,96,361,43,96,3
92,44,96,424,44,96
3160 DATA 456,45,96
```

```
1300 ▬ CROCODILO
1310 DATA 0,0,1,7,15,15,9,5,0,0
,128,192,248,255,127,95,1,3,6,1
2
1320 DATA 62,255,255,255,192,22
4,176,159,191,255,255,255
1330 DATA 0,0,0,0,0,248,252,255
,0,0,0,0,0,0,1,207,0,0,0,1,15,1
27,255,255
1340 DATA 0,3,63,255,255,255,25
5,255,127,255,255,255,255,254,2
49,247,248
1350 DATA 255,255,255,255,15,25
5,255
1360 DATA 0,224,254,255,255,255
,255,255,0,0,0,192,248,255,255,
255,0,2,4,7,7
1370 DATA 3,0,0,21,1,164,73,255
,255,0,0
1380 DATA 255,127,63,63,255,255
,127,31,255,255,255,255,255,255
,255,255
1390 DATA 239,239,239,239,239,2
47,247,247,60,255,255,255,255,2
55,255,255
1780 DATA REM ELEFANTE
1790 DATA 0,0,0,8,28,25,51,51,0
,0,0,126,255,193,253,253,0,0,0,
0,0,255,255
1800 DATA 255,0,0,0,0,0,248,252
,254
1810 DATA 102,111,111,111,125,5
7,26,0,253,251,251,251,231,31,1
5,15,255,255
1820 DATA 255,255,255,255,255,2
55,254,255,255,255,255,255,255,
255
1830 DATA 0,0,128,64,32,16,12,0
,15,15,15,14,14,14,14,31,255,24
0,224,224
1840 DATA 224,224,224,224,255,6
3,59,59,57,57,121,248
1850 DATA 0,0,128,192,224,224,1
28,0
1860 REM ARVORE 1
1870 DATA 0,0,0,0,1,1,3,7,0,0,0
,0,224,240,248,248,15,63,63,63,
31,15,3,3
1880 DATA 252,254,254,254,254,2
54,252,252
1890 DATA 3,3,3,3,3,1,0,0,248,2
48,248,248,248,248,240,96,96,96
,96,96,96,96
1900 DATA 96,96
1910 REM ARVORE ▮
1920 DATA 0,3,15,31,127,127,63,
1,7,15,255,255,255,255,255,255,
15,63,255,255
1930 DATA 255,255,255,255,0,128
,248,248,248,248,240,224
1940 DATA 255,227,96,48,24,25,1
3,15,252,240,96,96,192,192,128,
128,7,7,7,7,7
1950 DATA 7,7,7,7,7,7,15,15,15,
31,63
```

```
10 CLEAR 1000:CLS
20 DIM C(59),E(17),T1(1),T2(7),
F1(7),F2(7)
30 PMODE 3,1:PCLS
40 W$="L6UL6D":W$=W$+W$+W$+W$
50 DRAW"BM1,4C4RUR2UR2DR2DR4DR8
UR4UR2UR2UR2UR4DR2DR2DR4D2R6DR2
DR4DR4UR4UR4UR4UR4UR4UR4UR4UR10
DR6DR4DR2DR2DR2D15"+W$+W$+"L6UL
6DL4BU14DR11FRFR7FR3FR9FGL7GL21
HL2D3FR21FR3F"
60 PAINT(50,10),4
70 DRAW"BM98,5C1L8GLGLGD5BFD2RF
R3FBM32,2GFREHLBM1,8C2FRERFBR4U
BM+4,2;RBR3RBM+3,1;RBM+3,1;RBR3
RBL8BDLBL5NEBL6NEBL4EBL5E"
80 GET (0,0)-(112,20),C,G
230 PCLS
240 DRAW"BM4,0C2DG2DG2D3R2DE2UE
2UE43FRFR11F3RFRFR3DBL8NU3D4F2D
GH2UHL3D5LURU3HL6D5L2BU6L3D6NL2
U6LH2LNGUHL2"
250 PAINT(20,7),2
260 DRAW"BM12,3C3R2D4G2BM8,4R"
270 GET(0,0)-(37,17),E,G
280 PCLS 4
290 DRAW"BM7,19C1H3U5H5UE6R2F3D
F2D2G3D3G2D2":PAINT(7,7),1
300 DRAW"BM20,5E2R2E2RFR4E2F2R2
E2R5FRFDG3LGLGLGL5HL5H2L4":PAIN
T(34,5),1
310 GET(0,0)-(13,19),F1,G:GET(2
0,0)-(49,9),F2,G
```

```
320 PCLS:DRAW"BM0,0C4D20BE20F3D
FD15G2NL2R8HL4EU13E4U2"
330 GET(0,0)-(1,20),T1,G:GET(20
,0)-(31,22),T2,G
340 PCLS3:SCREEN 1,0
350 CIRCLE(255,191),160,1,.6:PA
INT(230,180),1
360 CIRCLE(0,191),140,2,.35,.75
,1:PAINT(10,180),1,2
380 PUT(206,100)-(243,117),E,PS
ET
390 FOR K=1 TO 10: READ X,Y:PUT
(X,Y)-(X+13,Y+19),F1,AND:PUT(X+
6,Y+20)-(X+7,Y+40),T1,OR:NEXT
400 FOR K=1 TO 10:READ X,Y:PUT(
X,Y)-(X+29,Y+9),F2,AND:PUT(X+8,
Y+9)-(X+19,Y+31),T2,OR:NEXT
410 COLOR 2:LINE (138,187)-(255
,187),PSET:PAINT(255,191),2:PAI
NT(255,191),3,1
420 PUT(143,166)-(255,186),C,PS
ET
450 DATA 16,110,24,113,34,108,4
8,110,56,108,190,80,198,82,212,
84,210,79,240,70
460 DATA 2,120,18,122,28,116,46
,118,60,124,160,90,174,95,190,9
0,214,86,226,90
470 GOTO 470
```

Executando estes programas, veremos o tipo de figura que podemos criar usando UDG. Não se preocupe se o alto da tela parece vazio no momento, pois ■ cenário ainda não está completo.

A água sob ■ crocodilo é uma boa demonstração da versatilidade dos blocos gráficos: ela é composta pela repetição de um só bloco. (Isto não vale para o TRS-Color, onde a matriz do crocodilo inclui ■ água.)

O cenário não é formado apenas por UDG: empregamos também comandos gráficos para desenhar ■ colinas. Embora com freqüência seja melhor usar blocos gráficos em vez de comandos gráficos, podemos obter ótimos resultados combinando as duas técnicas. O próximo artigo completará o desenho; portanto, grave o programa em fita.

COMO FUNCIONA

Se você não entendeu como ■ programa define e guarda os blocos, consulte o último artigo desta série.

O programa pode ser dividido em duas partes: uma que define os blocos e outra que os coloca na tela.

O Spectrum utiliza o comando **POKE** para colocar os valores das linhas **DATA** na tela; o MSX, por sua vez, usa **VPOKE**. No programa do TRS-Color não há linhas **DATA**: desenhamos as figuras com **DRAW** e, em seguida, guardamos os padrões em matrizes com **GET**.

Criados os blocos, ■ programa prossegue colocando-os nos locais adequados.

O Spectrum usa uma série de comandos **PRINT AT** para desenhar cada segmento de animal ■ árvore. Ele também emprega comandos locais de cor para colorir ■ figuras.

Um comando local de cor é aquele que se aplica somente à linha onde ocorre. Em geral, fazemos com que o pano de fundo — **PAPER** — seja 8, que ■ transparente. Isso evita que os blocos impressos na tela o danifiquem.

UM PANO DE FUNDO ■

O pano de fundo é definido pelas linhas 410 ■ 440. A primeira delas estabelece a cor, e as duas seguintes produzem o céu azul-claro e o chão verde. Para conseguir as diferentes cores de fundo, imprimem-se espaços com as cores escolhidas, por meio de dois laços **FOR...NEXT**.

Como existem várias árvores no cenário, elas são desenhadas com subrotinas (linhas 810 ■ 820). As variáveis x e y correspondem às coordenadas de comandos **PRINT AT**. Podemos facilmente acrescentar novas árvores ■ desenho, colocando mais coordenadas (veja, no artigo da página 501, como escolher as coordenadas adequadas) e comandos **GOSUB** nessas duas linhas.

O programa do MSX usa ■ comando **VPOKE** para colocar os blocos na tela gráfica. Comandos gráficos do BASIC também são empregados para desenhar o fundo da figura.

Inicialmente, ■ linha 5 protege o topo da memória para que ■ linhas 10 a 30 coloquem ali os padrões dos blocos. A linha 30 também seleciona as cores da tela; ■ 110 seleciona ■ tela. As linhas 120 a 140 desenham as duas colinas que aparecem ■ fundo.

Em seguida, usando **VPOKE**, os padrões são colocados na tela gráfica. A posição que o bloco deve ocupar na tela — A —, ■ posição do bloco no banco de blocos — ■ — e a cor do bloco — C — são obtidos com **READ** nas linhas **DATA** que começam em 3000.

Já que existem dois tipos de árvore, elas são desenhadas por dois laços **FOR...NEXT**. Precisamos, ■ entanto, de um grande número delas. Para fazê-las, usamos uma série de valores que ■ somam às posições em que cada árvore é desenhada. Esses valores são lidos nas linhas **DATA** 3115 ■ 3135, mudando a posição de cada árvore sempre que o laço é executado.

As cores não foram colocadas no banco, e sim nas linhas **DATA** a partir de 3000, juntamente com as posições dos blocos. Isso foi possível porque, em nosso exemplo, cada bloco tinha apenas duas cores. Quando o colorido dos blocos for mais complicado, convém usar ■ banco.

O programa do TRS-Color começa reservando espaço para os cordões que usa e **DIM**ensionando as matrizes que guardam os blocos gráficos. Com exceção das árvores, temos um bloco gráfico para cada figura, já que os blocos podem ter qualquer tamanho. As árvores requerem dois blocos cada, pois têm uma metade vermelha e outra verde.

OS ANIMAIS

O programa prossegue desenhando um animal de cada vez e guardando cada um deles na matriz apropriada, com o comando **GET**. A linha 80 guarda ■ crocodilo, a linha 270 o elefante, a linha 310 cuida das copas das árvores e ■ linha 330 dos troncos.

A segunda parte do programa coloca cada parte do desenho ■ seu lugar. As primeiras linhas limpam e selecionam a tela apropriada, desenhando também ■ colinas que fazem parte do cenário. As cores das colinas são determinadas pelo comando **PAINT**.

Os dois grupos de árvores, cada um em uma colina, são desenhados pelo laço **FOR...NEXT** das linhas 390 e 400, usando as linhas **DATA** 450 e 460 para determinar a posição.

As linhas 380 e 420 colocam o elefante e ■ crocodilo na tela, respectivamente. A última linha ativa do programa (se ignorarmos as linhas **DATA**) é a 470. Ela impede que o programa termine, permitindo a visualização da tela que, de outra forma, seria apagada. Use <**BREAK**> para parar o programa.

UMA MANADA DE ELEFANTES

Podemos substituir o elefante por uma manada. O método é ■ mesmo utilizado para as árvores: um laço **FOR...NEXT** ou vários comandos **GOSUB**.

Já vimos como usar blocos gráficos para escrever ■ tela de alta resolução.

Agora vamos examinar como carregar bytes na área reservada ao banco de blocos — segunda página de vídeo — por meio do monitor.

O monitor permite que coloquemos valores diretamente na memória do micro. Estes números serão, em nosso caso, os bytes correspondentes aos blocos gráficos necessários para o cenário.

Depois de colocar todos estes bytes na memória, carregue e execute o gerador de blocos, para poder ver, editar e entender como foram criadas as figuras. No próximo artigo apresentaremos o programa BASIC que desenha a floresta usando o banco de blocos que estamos criando hoje.

Para carregar o banco de blocos, ative o monitor com:

```
CALL - 151
```

ou **LM** no TK-2000.

Agora, copie a listagem. Para alterar o conteúdo de uma posição de memória, digite o endereço pretendido, seguido de dois pontos e dos valores desejados. Em nosso exemplo, modificamos oito posições de cada vez. Lembre-se de que todos os números precisam estar na forma hexadecimal. No TK-2000, devem-se usar outros endereços. Em vez de começar no endereço 4000, em hexa, use A000, também em hexa.

Note que nem todas as posições da área do banco precisam ser modificadas, havendo várias lacunas na listagem.

```
4008:  00 00 50 54 54 54 44 44
4010:  00 00 00 02 [illegible] 2A 2A 08
4018:  00 40 40 50 14 55 55 55
4020:  02 0A 0A 2A 28 2A 2A 2A
4028:  00 00 00 00 00 05 15 55
4030:  00 00 00 00 00 00 28 2A
4038:  00 00 00 40 50 54 55 55
4040:  00 20 2A 2A 2A 2A 2A 2A
4048:  54 55 55 55 55 55 05 51
4050:  2A 2A 2A 2A 2A 00 2A 2A
4058:  55 55 55 55 55 55 55 55
4060:  00 00 0A 2A 2A 2A 2A 2A
4068:  00 00 00 01 15 55 55 55
4070:  00 00 00 00 00 02 0A 2A
4078:  C0 D0 D4 D4 C4 C4 C4 [illegible]
4080:  AA 8A 82 80 80 80 80 80
4088:  80 80 80 [illegible] C0 C0 D0 D0
4090:  AA AA AA AA 8A 8A 82 82
4098:  81 81 81 81 80 80 80 80
40A8:  D0 C0 80 80 80 80 80 80
40B0:  8A 82 80 80 80 80 80 80
40C8:  C0 C0 90 90 84 84 81 81
40F0:  [illegible] 00 20 28 28 28 2A 2A
40F8:  40 54 55 55 55 55 55 55
4100:  2A 2A 2A 2A 2A 2A 2A 2A
4108:  55 55 55 55 55 55 55 55
4110:  2A 2A 2A 2A 2A 2A 2A 2A
4118:  55 55 55 55 55 55 55 55
4120:  2A 2A 2A 2A 2A 2A 2A 2A
4128:  01 15 55 55 55 55 55 55
4130:  00 00 02 0A 0A 0A 2A 2A
4148:  10 14 54 54 54 00 00 00
4150:  00 02 22 2A 2A 00 00 00
4158:  40 44 15 55 55 00 00 00
4160:  2A 2A 2A 2A 2A 28 20 20
4168:  55 55 55 55 55 55 55 55
4170:  2A 2A 2A 2A 2A 2A 2A 2A
4178:  55 55 55 55 55 55 55 55
4180:  2A 2A 2A 2A 2A 2A 2A 2A
4188:  51 51 51 51 51 45 45 55
4190:  2A 2A 2A 2A 2A 2A 2A 2A
4198:  55 55 55 55 55 55 55 55
41A0:  2A 2A 2A 2A 2A 2A 2A 2A
41A8:  55 55 55 55 55 55 55 55
41B0:  2A 2A 2A 2A 2A 2A 2A 2A
41C8:  D0 D0 D0 D0 D0 D4 D4 D4
41D0:  80 80 80 80 A0 A8 AA AA
41D8:  80 80 80 85 95 95 D5 D0
41E0:  80 80 80 80 80 80 80 82
41F8:  C0 D0 D0 D0 D0 D0 D0 D0
4200:  82 AA 82 82 82 80 80 80
4230:  2A 2A 2A 2A 2A 2A 2A 2A
4238:  55 55 55 55 55 55 55 55
4240:  2A 2A 2A 2A 2A 2A 2A 2A
4248:  55 55 55 55 55 55 55 55
4250:  2A 2A 2A 2A 2A 2A 2A 2A
4258:  55 55 55 55 55 55 55 55
4260:  2A 2A 2A 2A 2A 2A 2A 2A
4268:  55 55 55 55 55 55 55 55
4270:  2A 2A 2A 2A 2A 2A 2A 2A
4280:  AA D5 D5 D5 D5 D5 D5 D5
4288:  AA AA AA AA AA AA AA AA
4290:  AA D5 D5 D5 D5 D5 D5 D5
4298:  A5 AA AA AA AA AA AA AA
42A0:  AA D5 D5 D5 D5 D5 D5 D5
42A8:  A5 AA AA AA AA AA AA AA
42B0:  AA D5 D5 D5 D5 D5 D5 D5
42B8:  A5 AA AA AA AA AA AA AA
42C0:  AA D5 D5 D5 D5 D5 D5 D5
42C8:  A5 AA AA AA AA AA AA AA
42D0:  AA D5 D5 D5 D5 D5 D5 D5
42D8:  A5 AA AA AA AA AA AA AA
42E0:  AA D5 D5 D5 D5 D5 D5 D5
42E8:  A5 AA AA AA AA AA AA AA
42F0:  AA D5 D5 D5 D5 D5 D5 D5
4308:  C0 C0 C0 C0 C0 C0 C0 C0
4310:  AA AA 8A 8A 8A 8A 8A 8A
4318:  D0 D1 D1 D1 95 84 80 80
4320:  82 82 82 80 80 80 80 80
4338:  D0 D0 D0 D4 D4 D5 D5 D5
4340:  80 80 80 80 80 80 80 80
4370:  2A 2A 28 28 28 20 00 00
4378:  55 55 55 55 55 55 54 40
4380:  2A 2A 2A 2A 2A 2A 2A 2A
4388:  90 90 D0 D0 C0 C0 55 55
4390:  80 80 80 80 A2 A2 A2 AA
4398:  84 84 85 85 81 81 55 55
43A0:  2A 2A 2A 2A 2A 2A 2A 2A
43A8:  55 55 55 55 55 55 15 01
43B0:  2A 2A 0A 0A 0A 02 00 00
4448:  C0 C0 C0 C0 C0 C0 80 80
4450:  8A 8A 8A 8A 8A 8A AA AA
4470:  A0 A0 A8 A8 AA AA AA AA
4478:  95 85 85 81 81 80 80 80
44D0:  AA AA AA AA AA AA AA AA
4500:  00 00 00 00 00 00 40 50
4508:  00 00 00 00 00 00 7E 3A
4510:  00 00 00 00 00 00 01 15
4518:  00 00 00 00 00 60 68 7A
4520:  00 00 00 00 55 55 55 55
4528:  00 00 00 00 2A 2A 2E 2E
4530:  00 00 00 00 01 05 15 7D
4590:  AA AA A8 A8 A8 A8 A0 A0
4598:  80 81 81 81 81 85 85 95
45B0:  AA AA AA AA AA A8 A8 A8
45B8:  D0 D4 D4 D4 D0 C1 C1 D1
45C0:  A8 AA AA [illegible] [illegible] AA A8 A8
45C8:  85 95 95 95 95 91 95 95
4610:  AA AA AA AA AA AA AA AA
4640:  50 50 54 54 54 FD 7F 5D
4648:  3A 3A 2A 22 00 00 20 2A
4650:  55 45 45 45 45 14 14 50
4658:  2A 2A 6A 7E 6A 2A 2A 2A
4660:  55 55 05 01 01 01 05 55
4668:  2E 3F 2A 20 00 00 20 6A
4670:  FD 5D 55 05 50 55 5D 57
4678:  00 02 00 2A 2A 2E 2E 2E
4680:  00 00 00 55 7F 5D 5D 5D
4688:  00 00 00 02 2A 3A 3A 3A
4690:  00 00 50 14 15 55 55 55
4698:  20 2A 2A 2A 2A 3E 0E 0F
46A0:  01 05 05 05 04 [illegible] 04 00
46D8:  D5 D5 D4 D4 [illegible] D5 D5 D5
46E0:  80 AA AA AA AA AA AA AA
46E8:  [illegible] D5 D5 D5 D5 D5 D5 D5
46F0:  A8 AA AA AA AA AA AA AA
46F8:  D5 D5 D5 D5 D5 D5 D5 D5
4700:  AA 82 AA AA 82 80 80 80
4708:  95 95 D5 D5 D4 D4 94 80
4710:  80 A8 80 80 88 [illegible] A8 AA
4718:  [illegible] 81 81 81 81 81 81 81
4750:  AA AA AA AA AA AA AA AA
4780:  54 50 00 00 00 00 00 00
4788:  3A 0A 00 00 00 00 00 00
4790:  50 40 40 00 00 00 00 00
4798:  68 6A 62 2A 0A 2A 28 00
47A0:  5F 57 55 D5 55 00 55 55
47A8:  2B 2B 2B 2F 2A 00 2A 2A
47B0:  57 57 57 15 11 04 05 01
47B8:  3F 2B 02 00 00 00 00 00
47C0:  55 55 00 00 00 00 00 00
47C8:  7E 2A 00 00 00 00 00 00
47D0:  55 55 55 54 50 40 40 00
47D8:  0B 03 2A 03 02 02 0A 2A
47E0:  00 10 05 10 00 [illegible] 00 00
4810:  80 80 80 A0 A0 A0 A0 A0
4818:  D5 D5 [illegible] D5 D5 D5 D5 D5
4820:  AA AA AA AA AA AA AA AA
4828:  D5 D5 D5 D5 D5 D5 D5 D5
4830:  AA AA AA AA AA AA AA AA
4838:  D5 D5 D5 D5 D5 D5 D5 D5
4840:  82 8A 8A 8A 8A [illegible] 82 [illegible]
4848:  C0 C0 C0 D0 D0 D0 94 95
4850:  AA 8A [illegible] 82 82 80 80 [illegible]
4888:  00 00 00 00 C0 C0 C0 [illegible]
4890:  AA AA AA AA AA AA AA AA
4898:  00 00 00 00 81 81 81 85
4948:  80 [illegible] 80 80 C0 D0 D0 D4
4950:  A0 A0 A0 A8 AA AA AA AA
4958:  95 95 95 95 95 85 85 81
4960:  AA AA AA AA AA AA AA AA
4968:  81 81 81 81 81 81 81 81
4978:  D4 D0 D0 D0 D0 D0 D0 D0
4980:  82 82 82 A2 A2 AA AA AA
4988:  D5 95 95 85 85 81 81 80
```

Para gravar o banco na fita use:

```
4000.5000 W
```

No TK-2000 use:

```
A000.B000 W "nome"
```

Isto deve ser digitado ainda dentro do monitor. Quem tiver unidade de disquete pode utilizar o gerador para gravar o banco.

- O QUE SE PODE FAZER PARA PROTEGER PROGRAMAS BASIC
- PROGRAMAS AUTOCARREGÁVEIS: O QUE SÃO E O QUE FAZEM
- PROGRAMAS INTERDEPENDENTES
- COMO DESATIVAR OS COMANDOS **SAVE** E **LIST**
- PEQUENAS ARMADILHAS PARA O SEU COMPUTADOR

Nunca é demais dar a um programa um toque profissional, sobretudo depois de um grande esforço para deixá-lo "redondinho". Nesse sentido, você pode fazer duas coisas: melhorar a apresentação do programa e providenciar-lhe uma proteção razoável, de modo que suas técnicas especiais fiquem, ao menos, obscurecidas para o público.

Algum cuidado no acabamento é essencial se você pensa em comercializar seu programa — especialmente se pretende negociá-lo com uma *software-house*. Os recursos que você deve utilizar para dar-lhe uma melhor apresentação foram discutidos anteriormente; assim, focalizaremos aqui as técnicas disponíveis para garantir alguma proteção ao seu trabalho.

A proteção de um programa BASIC depende de armadilhas colocadas dentro do próprio programa. Os escritos em código de máquina podem receber um tratamento bem mais sofisticado.

Não existem restrições de nenhum tipo quanto ao emprego de armadilhas para evitar as cópias "piratas" ou a listagem de seu programa por curiosos. Quantas delas serão colocadas no programa é questão para você decidir.

De qualquer maneira, convém sempre se lembrar de que não se conhecem métodos absolutamente seguros de evitar que um programa seja copiado. Muitas pessoas encaram os programas protegidos como um desafio e não desistem até que ele seja vencido. Outras se empenharão em listar o programa para examiná-lo, adaptá-lo às suas necessidades ou, simplesmente, aprender.

Ainda que você tenha muita imaginação e domine as técnicas necessárias para montar as mais fantásticas armadilhas, seu programa nunca estará totalmente protegido dos olhares curiosos.
Mas, combinando determinados recursos — alguns mais simples, outros nem tanto —, você pode dificultar tanto o trabalho do "pirata" que ele se sentirá tentado a abandonar sua tarefa.

PRIMEIROS PASSOS

Um dos métodos para proteger programas consiste em introduzir neles várias armadilhas simples. Elas não impedem que alguém com conhecimento da máquina abra o programa, mas tornam a tarefa bem mais trabalhosa. Infelizmente, esse método dificulta também a elaboração do programa. Além disso, ele complica a depuração de erros, já que as armadilhas podem estar ativas durante a execução do programa.

As travas mais simples são as que introduzem no programa mudanças que tornam impossível o uso dos comandos **SAVE**, **LIST** e outros. Como elas só se ativam após a execução do programa, não criam os problemas de depuração mencionados acima.

AUTOCARREGAMENTO

Pode-se obter uma proteção bem melhor fazendo com que o programa seja executado automaticamente depois de carregado. No processo normal de carregar um programa (**LOAD**), os comandos são digitados no modo direto. Mas eles podem ser chamados por um programa à parte, denominado programa de autocarregamento (*bootstrap*). Este é escrito em BASIC ou código de máquina, dependendo das tarefas específicas que deve executar. Na sua forma mais simples, pode ser algo como:

```
10 LOAD "NOME PROXIMO PROGRAMA"
```

A execução desse pequeno programa carregaria para a memória do micro o programa cujo nome fosse especificado. Obviamente, o uso de uma linha como esta é bem específico. Mas esse tipo de programa pode ser empregado para fazer muito mais, sendo bastante útil para programação suplementar — como montar páginas-títulos, gráficos etc.

Estas incluem poderosas armas de proteção de um programa BASIC, que alteram determinados comandos do sistema. No nosso caso, o programa de auto-carregamento funciona como um inicializador do sistema que permite que seu programa seja carregado e executado.

Vários tipos de programas comerciais empregam *bootstraps*, muitas vezes gravados na forma de programas patrocinados, onde cada parte é carregada em seqüência. Lembre-se de que, normalmente, um programa BASIC apaga o anterior ao ser carregado. Assim, se você utilizar ■ técnica, carregue antes os módulos em linguagem de máquina e, depois, o BASIC.

Com programas patrocinados, a proteção pode ser determinada por um certo grau de interdependência entre um arquivo (parte do programa) ■ outro. Neste caso, um arquivo checa um valor de memória determinado por outro. Pode-se ainda acrescentar um programa que cheque um arquivo de dados especial colocado após o programa principal. Em ambas as alternativas, a falta de qualquer um dos módulos provoca um erro e impede que o programa funcione.

Os bootstraps oferecem mais uma vantagem: com sua utilização, o tempo de carregamento dos programas na memória diminui, porque ■ código de máquina e os dados podem ser colocados diretamente na memória. Esse programa dispensa, assim, o uso de declarações **DATA** do BASIC, que só funciona após o programa começar a ser executado.

AUTO-EXECUÇÃO

A utilização de um programa de autocarregamento não é muito simples no TRS-Color (usando-se o cassete). Esse computador não dispensaria chamadas especiais do sistema, o que não é possível com o BASIC. No Spectrum, porém, para executar um programa a partir de outro basta usar o comando apropriado em algum lugar do programa de autocarregamento.

```
■ LOAD "NOME DO PROXIMO ■
AMA"
```

Grave (**SAVE**) este segundo programa usando **SAVE "NOME DO PROXIMO PROGRAMA" LINE 1** — ou o número de linha que represente o início do programa. Escolhendo um número de linha maior, você poderá incluir avisos ou dados para serem lidos (**PEEK**) e checados em linhas REM anteriores à linha inicial de execução.

No MSX também é muito simples executar um programa a partir de outro: basta usar o comando **LOAD** com ■ opção R. Mas, para que tudo dê certo, o programa a ser executado deve estar gravado no formato ASCII, isto é, com a instrução **SAVE**. Na sua forma mais simples, ■ programa seria:

```
100 LOAD "NOME DO PROGRAMA",R
```

As técnicas que seguem valem apenas para micros com disquete.

Veremos, primeiro, como montar um programa de execução automática. O método mais simples consiste em introduzir no programa, que é automaticamente carregado e executado ao se ligar ■ computador, uma linha do tipo **PRINT CHR$(4): "RUN PROGRAMA"**. Mas ■ utilização de um arquivo-texto é mais elegante, e exige apenas que ■ programa com que ■ disquete foi inicializado contenha esta linha:

```
10 PRINT CHR$(4);"EXEC AUTORUN"
```

AUTORUN é um arquivo-texto que será executado como se as instruções estivessem sendo digitadas a partir do teclado. Para criá-lo, digite e execute este programa:

```
10 D$ = CHR$ (4)
20  PRINT D$;"OPEN AUTORUN"
30  PRINT D$;"WRITE AUTORUN"
40  PRINT "NOMON C,I,O"
50  PRINT "RUN BP35"
■   PRINT D$;"CLOSE"
```

Com esse arquivo, qualquer tentativa de interromper o programa será interpretada como um erro. ■ isso será aproveitado por uma rotina de tratamento de erros no programa principal.

Digite este programa e salve-o, por exemplo, com o nome de BP35:

```
10  HOME
■   PRINT "ALO    ";
30  GOTO 20
```

Em seguida, digite **"EXEC AUTORUN"**. O programa será executado pelos comandos do arquivo **AUTORUN**. Se inicializou o disquete como sugerimos, seu programa BP35 terá execução automática toda vez que o computador for ligado com esse disquete no drive 1.

O USO DE VARIÁVEIS DO SISTEMA

A auto-execução não basta para afastar os curiosos. É necessário utilizar outros recursos para evitar que ■ programas sejam interrompidos, listados ou alterados. Um desses recursos consiste em alterar o modo como o sistema reage quando se faz uma chamada específica — por exemplo, a sub-rotina de listagem (LIST).

Cada computador tem um sistema operacional e um interpretador BASIC. A maior parte da informação é guardada na memória apenas para leitura — ROM —, mas parte dela é transferida para a memória de acesso aleatório — RAM —, quando se liga o computador. Esta informação pode ser modificada para alterar o funcionamento do sistema, permitindo que se incorporem aos programas métodos sofisticados de proteção.

Como estaremos interferindo na própria forma de trabalho do computador, precisaremos ter em mãos uma listagem completa das variáveis e rotinas do sistema, e um bom mapa de memória. Veja o seu manual de referência.

A transferência de parte da informação do sistema da ROM para ■ RAM, possibilita a alteração do valor de algumas variáveis. Estando na RAM, esta informação torna-se vulnerável a qualquer modificação que se queira fazer.

Um tipo especial de variável do sistema é o apontador da RAM. Este consiste, em geral, de dois endereços consecutivos que guardam o endereço de uma sub-rotina específica. Se alterarmos tal endereço, o sistema será redirecionado sempre que esta sub-rotina for chamada. Os melhores apontadores para ■ proteção de programas são os que se referem aos comandos **LIST**, **SAVE**, **INTERRUPT** e **RESET**. Os últimos comandos fogem ao escopo deste artigo.

ESTUDO DE CASOS

É muito difícil dar total proteção aos programas, especialmente se se quer evitar um maior envolvimento com código de máquina. Os métodos para cada máquina diferem muito, já que ■ sistemas operacionais também são bastante diferentes. Mas vale a pena analisar algumas possibilidades.

Uma das maneiras mais simples de marcar um programa consiste em colo■ uma mensagem que não possa ser retirada. Para reforçar a proteção, convém acrescentar uma rotina que verifique a presença da marca.

Inicialmente, encontre o endereço da área reservada para programas em BASIC. Esse endereço (variável de sistema **PROG**) é armazenado nas posições 23635 e 23636 e pode ser determinado

pela instrução: **PRINT PEEK 23635 + 256 * PEEK 23636**. Conhecendo o valor de **PROG**, você pode colocar qualquer número em **PROG + 1**, alterando o número da primeira linha do programa.

O segredo é deixar a linha com o número 0, já que não é possível apagar a linha 0. Suponhamos que a primeira linha do programa seja:

```
10 REM (C) NOVA CULTURAL 1986
```

Digite este comando:

```
POKE(PEEK 23635 + 256 * PEEK
25636) + 1,0
```

Pronto! A linha 10 virou linha 0.

O mesmo pode ser feito pelo programa de autocarregamento. Mas, tratando-se de um programa auto-executável, a maneira óbvia de interrompê-lo é teclando <**BREAK**>, que coloca uma mensagem na parte de baixo da tela. Suponhamos, porém, que esta não aceite a mensagem. Na lista das variáveis de sistema, vê-se que **DFSZ** (no endereço 23659) armazena o número de linhas da parte inferior da tela (geralmente 2). Alterando este valor para 0, o computador detectará um erro assim que a mensagem tentar aparecer na tela.

Como não se pode entrar tais instruções no modo direto, digite:

```
10 POKE 23659,0
20 PRINT AT 5,5; ""
30 GOTO 20
```

Execute o programa. Se pressionar <**BREAK**>, provocará um erro.

Quando usar este método, lembre-se de que, se seu programa contiver situações que produzem mensagens como **INPUT?** ou **scroll?**, o erro será inevitável. Assim, para entrada de dados, use **INKEY$**.

O "pirata" pode evitar que um programa se auto-execute, carregando-o com **MERGE**. Mas será impossível fazê-lo se o programa for gravado em código. As variáveis do sistema, o programa e a memória livre devem ser gravados acima do buffer da impressora.

A gravação começa com **CODE 23552**, que é o início da área das variáveis do sistema. O número de bytes é N-23552, onde N é qualquer número maior que **STKEND** (o endereço do início do espaço livre). Obtém-se este endereço com **PEEK 23653 + 256*PEEK 23654**.

Coloque estas linhas no início do seu programa:

```
1 SAVE "NOMEPROG" CODE 23552,
  N-23552
2 POKE 23659,0
```

Agora, iguale N ao endereço descrito acima e digite **GOTO 1**. O programa será gravado. O comando **LOAD "NOMEPROG" CODE** fará com que o programa seja executado.

Vimos como proceder para que um programa inicial carregue e execute o programa principal. Mas isso não é suficiente. Um simples <**CTRL**><**STOP**> permite que se interrompa o programa para que ele seja listado.

No MSX, pode-se evitar uma interrupção desse tipo com facilidade. A idéia é direcionar o programa para uma sub-rotina especial toda vez que se tentar uma interrupção com <**CTRL**><**STOP**>. Para isso, utilizam-se as instruções **STOP ON**, que fazem com que o BASIC verifique a todo instante se uma interrupção foi tentada, e **ON STOP GOSUB XXXXX**, que desvia o programa para uma determinada linha em caso positivo. Digite o seguinte:

```
10 ON STOP GOSUB 1000
20 STOP ON
30 CLS
40 LOCATE 10,12:PRINTRND(1):GOT
O 40
1000 P=P+1
1010 CLS:LOCATE 10,10:PRINT"Não
 seja curioso!":BEEP:FOR S=1 TO
 1000:NEXT
1020 IF P=3 THEN NEW:END
1030 CLS:RETURN
```

Não se esqueça de gravar o programa antes de executá-lo. Depois, pare-o se puder!

Você já sabe como proceder para que seu programa seja automaticamente executado; veja agora como impedir que o interrompam, ou seja, como impossibilitar sua listagem. Para isso, fazemos com que toda tentativa de interrupção — por meio do <**CTRL**><**C**> ou do <**CTRL**><**RESET**> — seja interpretada como um erro, desviando a execução para uma rotina especial. Para que o micro interprete o <**CTRL**><**RESET**> como um erro (o <**CTRL**><**C**> é normalmente interpretado como tal), adicione as linhas seguintes ao programa do arquivo **AUTORUN**.

```
45  PRINT "POKE 40286,35"
46  PRINT "POKE 40287,216"
```

Elimine o **AUTORUN** já existente e execute o programa. O novo arquivo contém os dois **POKE** que mudam o comportamento do computador em relação ao <**RESET**>. Mas, para que possamos tirar proveito dessa mudança, uma rotina de erros deve ser inserida no programa principal:

```
5  ONERR  GOTO 1000
1000 P = P + 1
1010  IF P = 3 THEN  PR# 6
1020  PRINT  CHR$ (7);: RESUME
 NEXT
```

Grave o programa alterado. Desligue o computador e ligue-o novamente. Tente parar o programa...

Pode ocorrer, porém, que o curioso se lembre de olhar para o diretório do disco e carregar o seu programa diretamente, sem executá-lo; assim, poderá listá-lo. Para confundi-lo, inclua no nome do programa, no momento de gravar (**SAVE**), alguns caracteres de controle, que ficarão invisíveis. Para fazer isto, basta digitar junto do nome do programa um ou mais <**CTRL**><**letra**>, onde a letra pode ser A, B etc. Evite o C e o X.

ZX-81: EDIÇÃO DE PROGRAMAS

- COMO EDITAR LINHAS DE UM PROGRAMA BASIC
- OS MOVIMENTOS DO CURSOR
- O COMANDO EDIT
- CONSOLIDE AS MODIFICAÇÕES

Os micros da linha ZX-81 possuem bons recursos de edição. Com ■ comando EDIT e ■ teclas de movimentação do cursor, você pode alterar ou duplicar rapidamente uma linha de programa.

Devido à sua natureza "conversacional", todos os interpretadores da linguagem BASIC incluem alguns recursos de edição de programas, ou seja, técnicas de entrada e alteração das linhas que o compõem.

Como vimos em artigos anteriores sobre edição de programas em outras linhas de microcomputadores, o primeiro interpretador BASIC, desenvolvido na década de 60 na Universidade de Dartmouth, nos EUA, fixou os recursos mais elementares de edição: numeração, inserção, apagamento ■ listagem de linhas. Estes recursos, adotados posteriormente em todos os dialetos BASIC que surgiram, não davam ao programador ■ possibilidade de alterar caracteres. Assim, para corrigir um único caractere errado, ele teria que digitar toda a linha novamente.

Para poupá-lo desse trabalho, desenvolveram-se comandos como o **EDIT**, presentes nos micros da linha Sinclair, TRS-80 ■ TRS-Color. O comando **EDIT** dos micros da linha ZX-81 opera em uma linha de cada vez.

DIGITAÇÃO DE PROGRAMAS

No ZX-81, cada linha é digitada de uma vez, precedida de um número.

A parte inferior da tela do ZX-81 destina-se à digitação de linhas. Ao ser ligado, o computador exibe um cursor de texto — um retângulo em vídeo inverso — contendo ■ letra K. Esta é a chamada *linha de edição* da tela. À medida que você vai digitando ■ linha, por meio do teclado, este cursor vai se deslocando à frente dos caracteres. Durante o deslocamento, ele pode mudar de tipo (passar para tipo L, tipo G, tipo F etc.).

Quando se pressiona ■ tecla **<ENTER>** ou **<RETURN>**, ■ linha é completada e inserida no ponto certo do programa. Enquanto isso não ocorre, pode-se modificar ■ linha à vontade.

Três teclas de controle são utilizadas para se proceder às modificações: as teclas com ■ flechas para a esquerda e para ■ direita (← e →), que devem ser acionadas simultaneamente com a tecla **<SHIFT>**, e ■ tecla de apagamento (chamada de **DELETE** ou **RUBOUT**, conforme a marca do computador, e que, no ZX-81, corresponde à pressão simultânea das teclas 9 e **<SHIFT>**).

Ao ser acionada, a tecla de apagamento pode eliminar um caractere ou ■ palavra-chave do BASIC de uma vez. O restante da linha se desloca para a esquerda, de modo a preencher os claros deixados.

O ZX-81 dispõe do modo de inserção automático, ou seja, se digitarmos caracteres quando o cursor estiver no meio de uma linha, estes serão inseridos, por intermédio do deslocamento do restante da linha para a direita. Para sobrepor caracteres (escrever sobre caracteres já existentes), é necessário primeiro apagá-los.

O COMANDO EDIT

Uma vez pressionada a tecla **<ENTER>**, ■ linha digitada passa a fazer parte do programa. Se, depois disso, você digitar apenas o seu número, e pressionar **<ENTER>** de novo, ela será apagada. Por outro lado, se você digitar seu número, seguido de novos comandos ou instruções, estes substituirão a linha existente no programa.

Um recurso de edição que faz falta no ZX-81 é o comando **DELETE** (ou, ainda, **DEL**, em alguns computadores), que apaga grupos de linhas de uma vez só. Os usuários desse micro precisam sempre digitar os números de cada uma das linhas ■ serem apagadas, ■ a um, seguidos de **<ENTER>**.

Para modificar uma linha já existente, sem precisar redigitá-la inteiramente, pode-se recorrer ■ comando **EDIT**. Para isso é necessário entender o conceito de *cursor de linhas*.

Se você olhar para uma listagem de programa na tela, perceberá que existe sempre um retângulo em vídeo inverso, contendo o sinal de maior (>) em seu interior, e que aponta para uma das linhas na tela. Esse retângulo é o cursor de linhas. Para deslocá-lo, basta pressionar uma ou mais vezes uma das teclas com a flecha para cima ou para baixo (↑ ou ↓).

Agora, se você pressionar simultaneamente as teclas **<SHIFT>** e **1**, a linha apontada pelo cursor de linhas aparecerá ■ linha de edição.

Você poderá usar, então, todos os procedimentos de edição: movimentação do cursor de texto, apagamento, inserção etc. Tudo funciona da mesma maneira, até que você pressione a tecla **<ENTER>**. Ao fazê-lo, a linha modificada irá substituir a que foi chamada anteriormente pelo **EDIT**.

DUPLICAÇÃO DE LINHAS

A duplicação de linhas é um dos recursos mais interessantes do ZX-81. O número da linha pode ser editado normalmente. Assim, se apenas ele for modificado, a linha com ■ número original mantém-se no programa, e a mesma linha, com nova numeração, ■ inserida no ponto correto.

Por outro lado, se, além do número da linha, também ■ seu conteúdo for modificado, uma linha diferente será automaticamente gerada.

Portanto, para que ■ linha possa ser editada pelo comando **EDIT**, ela precisa estar listada na tela (use ■ comando **LIST número**, para isto). Desloque o cursor de linha até ela, ■ pressione o comando **EDIT**.

Para agilizar o processo de edição de várias linhas de ■ programa extenso, você pode utilizar um pequeno truque. Em vez de pressionar repetidas vezes as teclas de controle do cursor de linhas, até chegar ■ linha que você deseja, digite ■ comando **LIST**, seguido do número dessa linha. Isto a colocará no alto da tela, com o cursor de linhas automaticamente apontado para ela. Basta, então, pressionar o comando **EDIT**.

SÍMBOLOS GRÁFICOS NO MSX

- SÍMBOLOS GRÁFICOS
- ENTRADA PELO TECLADO
- AS FUNÇÕES CHR$ E STRING$
- TABELA DE REFERÊNCIA

Os micros da linha MSX dispõem de caracteres gráficos que podem ■ entrados diretamente pelo teclado ou usados dentro de um programa. Veja como explorar este recurso especial.

Os microcomputadores da linha MSX são extremamente versáteis do ponto de vista da programação gráfica. Em artigos anteriores, vimos como as teclas gráficas (**SCREEN**) podem ser programadas, em média e alta resolução, por meio de instruções poderosas como **LINE**, **CIRCLE**, **PAINT**, **PSET** etc.

O MSX apresenta, ainda, um recur■ gráfico adicional que é pouco explorado. São os *caracteres gráficos*, disponíveis para o programador por meio de dois procedimentos: entrada direta pelo teclado ■ inserção através das funções do BASIC **CHR$** e **STRING$**.

O que são caracteres gráficos? Como você já sabe, os caracteres que aparecem no vídeo têm códigos numéricos inteiros, entre 0 e 255; cada caractere corresponde, portanto, a um byte da memória do vídeo. Parte dessa codificação está convencionada internacionalmente pelo chamado código ASCII, que vai de 32 ■ 126. Os códigos de 0 a 31 são normalmente utilizados para funções de controle do vídeo, ■ dependem do tipo de computador. O mesmo acontece com os códigos de 127 ■ 255. Nesta faixa, cada fabricante usou os códigos de uma maneira, em geral para acomodar caracteres ■ outros idiomas que não o inglês (é o caso do MSX, existente no Brasil) ou para especificar caracteres gráficos. Estes tanto podem ser tipos especiais, para a imagem de um rosto sorrindo ou uma nota musical, como blocos gráficos formando linhas, ângulos e cantos.

GRÁFICOS DA ROM

Tais caracteres são também chamados de *gráficos da ROM*, pois já vêm pré-programados. No MSX, os caracteres especiais ocupam duas faixas da tabela de caracteres:

- ■ faixa de 1 a 31
- a faixa de 144 a 254

Nestas faixas, os caracteres gráficos propriamente ditos encontram-se dispersos ■ vários pontos da tabela, entremeados com caracteres de outros idiomas, com o alfabeto grego, símbolos matemáticos etc.

Os caracteres gráficos que mais nos interessarão neste artigo são os blocos empregados na formatação de tabelas, de formulários de entrada ou na composição de quaisquer outros desenhos formados por linhas retas. A utilização de caracteres gráficos, nesses casos, tem ■ vantagem de não complicar a programação, misturando tela gráfica com texto, o que ■ permite recorrer a comandos mais simples, como o **LOCATE**, o **PRINT**, ■ **INPUT** etc.

ENTRADA PELO TECLADO

Assim como os caracteres convencionais do ASCII (demarcados sobre as teclas do microcomputador), os caracteres especiais ■ gráficos podem ser entrados pelo teclado. Para isso, pressiona-se simultaneamente uma ou mais de três teclas de função — **<GRAPH>**, **<CODE>** ou **<SHIFT>**—, com outra tecla normal do teclado. Com isso, o caractere desejado aparece diretamente na tela (por exemplo, dentro de uma cadeia alfanumérica).

Com o programa abaixo, você verá como entrar caracteres pelo teclado. Ele desenha ■ tabela simples na tela, usando blocos gráficos, e depois coloca os nomes digitados pelo usuário nas linhas da tabela.

```
200 CLS
210 PRINT" ┌─────┬──────────────┐"
220 PRINT" │No.  │  Nome        │"
230 PRINT" ├─────┼──────────────┤"
240 FOR I=1 TO ■
250 PRINT" │     │              │"
260 PRINT" ├─────┼──────────────┤"
270 NEXT I
280 PRINT" │     │              │"
290 PRINT" └─────┴──────────────┘"
300 FOR I=1 TO ■
310 LOCATE 0,21:PRINT STRING$(3
0,32)
320 LOCATE 0,21:LINE INPUT "NOM
E: ";N$
330 LOCATE 1,(I*2)+1:PRINT USIN
■ "##";I;
340 LOCATE 7,(I*2)+1:PRINT LEFT
$(N$,10);
350 NEXT I
360 LOCATE 0,21:PRINT "FIM
 ":END
```

Os traços são teclados na seguinte seqüência:

Linha 210: **<GRAPH>** ■
<GRAPH> - (4 vezes)
<GRAPH> T
<GRAPH> - (11 vezes)
<GRAPH> Y
Linha 220: **<SHIFT>** **<GRAPH>**
Linha 230: **<GRAPH>** F
<GRAPH> - (4 vezes)
<GRAPH> G
<GRAPH> - (11 vezes)
<GRAPH> H

Linha 250: como ■ linha 220
Linha 260: como a linha 230
Linha 280: como a linha 220

Linha 290: **<GRAPH>** V
<GRAPH> - (4 vezes)
<GRAPH> B
<GRAPH> - (11 vezes)
<GRAPH> N

Há duas desvantagens nesta maneira de entrar caracteres gráficos. Primeiro, as teclas não têm qualquer marcação que auxilie o usuário a achar o gráfico correto; assim, é preciso consultar o manual, o que torna o processo bastante trabalhoso. Em segundo lugar, o programa não pode ser listado em uma impressora não gráfica, ou que não seja própria para o MSX.

■■RACTERES GRÁFICOS NO PROGRAMA

Para especificar caracteres gráficos dentro de um programa, sem precisar digitá-los diretamente, podemos usar as funções **CHR$** e **STRING$**: com o número de código do caractere desejado, elas permitem que os caracteres normais, especiais ■ gráficos com códigos na faixa de 32 ■ 254 sejam impressos na tela a partir de um programa. Por exem-

CARACTERES GRÁFICOS DO MSX

Código	Teclas	Caractere	Código	Teclas	Caractere
1	GRAPH '	⊖	192	GRAPH U	▭
2	SHIFT GRAPH '	■	193	SHIFT GRAPH D	▚
3	SHIFT GRAPH ´	■	194	GRAPH I	▄
[illegible]	SHIFT GRAPH C	♦	195	SHIFT GRAPH O	▔
5	GRAPH ´	♣	196	GRAPH A	⊡
6	GRAPH C	▪	197	SHIFT GRAPH I	▀
8	SHIFT GRAPH 9	◘	198	GRAPH J	▏
9	GRAPH 0	○	199	GRAPH D	▞
10	SHIFT GRAPH 0	◙	200	GRAPH L	▐
11	GRAPH ■	♂	201	SHIFT GRAPH L	▕
12	SHIFT GRAPH M	▮	202	SHIFT GRAPH J	▌
13	GRAPH '	♪	203	SHIFT GRAPH Q	▨
14	SHIFT GRAPH '	♫	204	GRAPH Q	▩
15	GRAPH Z	☼	205	GRAPH E	◸
16	SHIFT GRAPH G 13	†	206	SHIFT GRAPH E	◿
17	GRAPH B	┴	207	GRAPH W	▶
18	GRAPH T	┬	208	SHIFT GRAPH W	◀
19	GRAPH H	┤	209	SHIFT GRAPH S	⊠
20	GRAPH F	├	210	GRAPH S	⧓
21	GRAPH G	┼	211	SHIFT GRAPH N	▘
22	SHIFT GRAPH \	│	212	SHIFT GRAPH F	▗
23	GRAPH -	─	213	SHIFT GRAPH V	▝
24	GRAPH ■	┌	214	SHIFT GRAPH H	▖
25	GRAPH Y	┐	215	SHIFT GRAPH P	▒
26	GRAPH V	└	219	GRAPH P	█
27	GRAPH N	┘	220	GRAPH O	▄
28	GRAPH X	X	221	GRAPH K	▌
29	GRAPH /	/	222	SHIFT GRAPH K	▐
30	GRAPH \	\	223	SHIFT GRAPH U	▀
31	SHIFT GRAPH -	✢	248	SHIFT GRAPH Z	○
169	SHIFT GRAPH R	┌	249	SHIFT GRAPH C	▪
170	SHIFT GRAPH Y	┐	250	SHIFT GRAPH X	▪
188	SHIFT GRAPH C	◊	254	SHIFT GRAPH A	⊡

O repertório de caracteres gráficos do MSX é muito variado: inclui desde tipos especiais, ■■ uma nota musical ou um rosto sorrindo, até blocos gráficos compondo linhas, ângulos e cantos. Para entrá-los pelo teclado, oriente-se pelo quadro de referência ■ lado. Como você pode observar, será sempre necessário pressionar, simultaneamente, uma ou mais teclas de função e uma tecla normal.

plo, para imprimir na tela a letra grega beta, digite:

```
PRINT CHR$(225)
```

Ou, então, para traçar uma linha reta de dupla espessura [illegible] tela, use:

```
PRINT STRING$(32,197)
```

A função **STRING$ (n,c)** retorna uma cadeia de n caracteres com código c. Já os caracteres gráficos que se encontram na faixa de 1 [illegible] 31 não podem [illegible] impressos da mesma maneira, pois estes códigos têm funções de controle do vídeo. Para usá-los com [illegible] função **CHR$**, é necessário "avisar" o computador que [illegible] código será usado como gráfico. Para isso, são empregados dois bytes: **CHR$(1)**, seguido de **CHR$(n+64)**, onde n é o código do caractere gráfico na tabela. Por exemplo, **PRINT CHR$(1);CHR$(75)** mostrará na tela o símbolo masculino.

Esse método funciona também para a faixa de códigos gráficos que vai até 191. O programa abaixo mostra na tela uma tabela de correspondência:

```
10 KEY OFF:CLS:I=0
20 FOR N=1 TO [illegible]
30 I=I+1:IF I>191 THEN GOSUB 10
0:END
40 PRINT USING "### ";I;
50 PRINT CHR$(1)+CHR$(I+64);" "
;
60 NEXT N
70 PRINT:PRINT
80 L=L+1:IF L<10 THEN 20
90 GOSUB 100:L=0:GOTO 20
100 LOCATE 0,22
110 PRINT "Pressione qualquer t
ecla p/continuar"
120 IF INKEY$="" THEN 120
130 CLS:RETURN
```

Se você substituir três linhas — 10, 30 e 50 —, o programa servirá também para mostrar os códigos gráficos de 128 a 254.

```
10 KEY OFF:CLS:I=127
30 I=I+1:IF I>254 THEN GOSUB
   100:END
50 PRINT CHR$(I);" ";
```

Caso pretenda utilizar um caractere gráfico da faixa de 1 a 31 com freqüência num programa, armazene-o em uma variável alfanumérica, como mostra o exemplo abaixo. Os quatro símbolos dos naipes do baralho são armazenados em **N$** (e seus nomes em **E$**):

```
10 CLS
20 FOR I=1 TO [illegible]
30 READ E$(I)
40 N$(I)=CHR$(1)+CHR$(I+66)
50 PRINT N$(I),E$(I)
60 NEXT I
70 DATA Copas,Ouros,Paus,Espada
s
```

EFEITOS SONOROS NO SPECTRUM

Os recursos sonoros do Spectrum são limitados, [illegible] considerarmos apenas o comando **SOUND**. Porém, com código de máquina, poderemos produzir vários sons, [illegible] sirene a tiros [illegible] laser.

Em geral, [illegible] endereçamento da memória é uma função do microprocessador. Na maioria dos computadores domésticos, quando queremos usar um aparelho externo — como uma impressora, um televisor ou mesmo o teclado do computador — temos que fazê-lo por meio de uma posição de memória que está ligada a uma porta de saída. Porém, com micros que usam [illegible] Z-80, como [illegible] Spectrum, por exemplo, pode-se ter acesso direto às portas, tanto em BASIC, com os comandos [illegible] e **OUT**, como em código de máquina, com **in** e **out**.

[illegible]

Uma porta é um canal de comunicação entre o microprocessador e [illegible] mundo externo. Este inclui o teclado e tudo [illegible] que é periférico ao microprocessador, à RAM [illegible] à ROM.

Já tivemos oportunidade de ver como **in** [illegible] usado para dar acesso ao teclado. Os comandos **OUT** e **out** funcionam de modo semelhante, só que, em vez de receber dados, enviam-nos aos periféricos. Ambos têm acesso bem mais versátil [illegible] alto-falante do que o comando BASIC **SOUND** [illegible] podem ser usados, também, para controlar [illegible] moldura da tela. Entretanto, como o **OUT** [illegible] excessivamente lento, consideraremos apenas o comando **out**.

Para gerar sons com **out** precisamos da velocidade da linguagem de máquina. Eles são obtidos por meio do movimento do cone do alto-falante para fora e para dentro. Movimentando-o uma vez, produzimos um clique, do tipo que [illegible] ouve ao ligar um aparelho de som. O truque está [illegible] repetir o movimento muitas vezes, e rapidamente.

Teremos, assim, uma sucessão de cliques, que provocam um efeito similar a um zumbido. Quanto mais veloz for o movimento de vaivém do cone, mais agudo será o som resultante.

A rotina Assembler apresentada a seguir produz um som cuja tonalidade vai aumentando. Digite **CLEAR 64599** e depois [illegible] linhas:

```
10 [illegible] org 64600
20 REM ld a,(23624)
30 REM rrca
```

```
 40 REM rrca
 50 REM rrca
 60 REM ld b,0
 70 REM loop push bc
 80 REM xor 16
 90 REM out 254,a
100 REM pause nop
110 REM nop
120 REM djnz pause
130 REM pop bc
140 REM djnz loop
150 REM ret
```

A porta 254, que controla o alto-falante, controla também a moldura da tela. Para que um comando **out** não modifique a moldura ao produzir um som, utiliza-se uma rotina em código destinada a evitar o problema.

Inicialmente, coloca-se no acumulador o valor da variável do sistema situado na posição 23624 da memória. As cores do Spectrum têm códigos que vão de 0 a 7. Normalmente, os bits que controlam a cor são os bits três, quatro e cinco. Contudo, quando estamos controlando a moldura via porta 254, os bits zero, um e dois encarregam-se da modificação das cores.

Assim, para se ter a certeza de que a cor da moldura não será modificada pelo efeito sonoro, os bits três, quatro e cinco devem ser deslocados três posições para a direita (para o lugar dos bits zero, um e dois).

SHIFT E ROTATE

O deslocamento dos bits pode ser executado por vários comandos. Utilizamos aqui **rrca** (*rotate right with carry, on the acumulator*, expressão em inglês que significa rotação para a direita com "vai um" no acumulador). Essa instrução movimenta todos os bits que estão no acumulador uma posição para a direita. É chamada de rotação porque coloca o conteúdo do bit zero no bit sete, fazendo-o "dar a volta", por assim dizer. Um comando SHIFT — deslocamento — moveria todo o conteúdo do acumulador um bit para a direita, mas preencheria o bit de reserva com zero. Seu emprego é mais adequado a operações aritméticas: um SHIFT para a esquerda multiplica um número por dois, enquanto um SHIFT para a direita o divide por dois.

Aqui, porém, utilizamos uma rotação porque queremos deslocar apenas três dos bits. O conteúdo dos demais não nos interessa. O comando **rrca** ainda copia o conteúdo do bit zero no bit carry — "vai um". Mais uma vez, não importa o valor do carry.

Para deslocar os bits de cor da moldura três posições para a direita, emprega-se **rrca** três vezes, o que equivale a dividir a cor da moldura por oito. Mas quando a cor — normalmente um número entre 0 e 7 — se encontrava nos bits três, quatro e cinco, estava multiplicada por oito.

Agora, quando usarmos o comando **out**, ele especificará a mesma cor de moldura que havia antes, e nenhuma mudança ocorrerá.

ACERTE OS CONTADORES

O registro B funciona como um contador duplo. Para começar, colocaremos 0 nele, e este valor será guardado na pilha por **push bc**. O registro B não pode ser guardado sozinho na pilha, já que os comandos de **push** e **pop** agem apenas em pares de registros. Precisamos, assim, guardar B juntamente com C. Como não usaremos o registro C, isso não afetará o programa.

COMO PRODUZIR SONS

O bit quatro da porta 254 controla o alto-falante. Alternando o valor desse bit, movimenta-se o cone do alto-falante, produzindo som.

Alterna-se o bit quatro por meio da operação lógica "ou exclusivo". Fazemos um "ou exclusivo" entre o valor do acumulador e dezesseis. Assim, se o bit quatro estiver ligado, valendo 1, ele será desligado, passando a valer 0 — ou vice-versa.

O **out 254,a** envia o conteúdo do acumulador via porta 254. Em muitos Assemblers comerciais os comandos **out** e **in** precisam de colchetes envolvendo o número da porta.

ACERTE O TOM

A instrução **nop**, com ■ código hex 0 0, significa "sem operação" — **no operation**. Ela não faz absolutamente nada, mas, como leva um certo tempo para ser executada — aproximadamente um microssegundo, ou seja, um milionésimo de segundo —, atrasa o microprocessador ■ realização do laço. Por ■ razão, ■ usada duas vezes neste programa. A velocidade com que o microprocessador executa o laço controla ■ freqüência de vibração do cone do alto-falante e, conseqüentemente, a tonalidade do som produzido.

Como você pode observar, os **nop** não são executados apenas duas vezes. A instrução **djnz** (decrementa e salta se não for zero) promove várias repetições da pausa.

A instrução **djnz** opera com base no valor do registro B. Assim, na primeira execução do laço, ela decrementa o valor de B, que passa de 0 para 255. Em seguida, o laço é executado mais 255 vezes, até que ■ valha 0 novamente.

Uma vez concluído o laço, o valor no topo da pilha é recuperado com **pop** e colocado no par BC. Com ■ recuperação do valor original de B, **djnz** o decrementa e o microprocessador entra mais ■ vez no laço. Esse processo leva o valor de BC de volta à pilha. Assim, o contador que fica na pilha é decrementado cada vez que o microprocessador executa o laço, e o valor inicial de B, que determina o número de execuções, torna-se menor. À medida que o número de execuções da pausa diminui, ■ freqüência do som aumenta.

Quantas portas existem?

Teoricamente, existem 64K portas — este é o maior número que pode ■ endereçado por um par de registros de oito bits. Na prática, porém, utilizam-se apenas umas poucas portas. Quando seu Spectrum está na configuração original, somente ■ porta 254 é usada.

O comando **In** permite empregar diferentes parâmetros para dirigir a porta para as várias regiões do teclado. Neste artigo, recorremos a bits distintos, da mesma porta, para controlar periféricos diversos.

Se seu Spectrum for conectado a periféricos não usuais, talvez você tenha ■ possibilidade de aproveitar outras portas. Conhecendo razoavelmente eletrônica, poderá, por exemplo, ligar seu micro ■ um sistema de aquecimento ou ■ um sintetizador, utilizando portas diferentes.

MOLDURA ■ SETE ■

Para especificar a cor da moldura utilizando o comando BASIC **BORDER**, atribuímos ■ ela um valor qualquer entre 0 ■ 7. Toda a moldura fica, então, com ■ cor correspondente.

A rotina seguinte ■ código de máquina para criar uma moldura com sete cores. Poderíamos trabalhar com oito cores, mas não faria sentido ter na moldura uma tonalidade idêntica à da tela principal.

```
 10 REM org 64600
 20 REM redo halt
 30 REM xor a
 40 REM loop out 254,a
 50 ■ ld b,205
 60 REM pause ld e,2
 70 REM inner dec e
 80 REM jr nz,inner
 90 REM djnz pause
100 REM ld d,a
110 REM ld a,$7F
120 REM in a,254
130 REM rra
140 REM rts nc
150 REM ld a,d
160 REM inc a
170 REM cp 7
180 REM jr nz,loop
190 REM jr redo
```

■ A TELA

A instrução **halt** aguarda a ocorrência de uma interrupção para, depois, continuar ■ programa. No Spectrum, a interrupção ocorre ■ cada varredura da tela. Assim, **halt** inicia a rotina quando a varredura começa ■ alto da tela, sincronizando ■ posição da moldura com a borda superior da tela de TV.

A instrução ■ ■ executa um "ou exclusivo" entre o acumulador e ele mesmo, ■ que constitui uma forma rápida de zerar ■ acumulador. O 0 é enviado pela porta 254. Como zero, ■ linguagem de máquina e em BASIC, corres-

ponde ■ preto, ■ porçao superior da tela adquire esta cor.

AJUSTE A PAUSA

Nesta rotina ■ tempo é essencial. O comprimento da pausa determina ■ largura das bandas coloridas da moldura. Controla-se a pausa por meio de dois laços: o interno, que acerta o tempo grosseiramente; e o externo, que o sintoniza de modo mais refinado.

O laço interno ■ executado com base no registro E, que recebe o valor ■ através de ■ **e,2**, sendo decrementado por **dec e**. A instrução **jrnz,inner** realiza um salto relativo em direção ao rótulo **inner**, se o resultado não for zero. Assim, a cada execução do laço externo, ■ laço interno é executado duas vezes.

O laço externo repete-se 205 vezes com o registro B. **ld b,205** coloca este valor no registro B, assim como o **djnz** que ■ decrementa, saltando enquanto o resultado não for zero. Nenhuma instrução deste tipo opera com o registro E; por causa disso, ■ subtração e o salto precisam ser feitos em duas instruções separadas.

Se alterarmos tais valores, veremos que ■ valor colocado em E modifica bastante a largura das bandas, enquanto o valor colocado no registro B tem um efeito bem menor.

A [illegible] DE UM BREAK

Certamente, em algum momento, desejaremos sair desse laço. Para fazê-lo sem desligar seu micro, programe uma rotina de saída.

A que apresentamos aqui verifica se ■ tecla <**BREAK**> foi pressionada, por meio da instrução **in**.

Inicialmente, porém, **ld d,a** coloca o valor do acumulador no registro D. Por um instante ■ acumulador será usado para outras coisas, ■ ■ registro D ficará desocupado, podendo servir de registro temporário.

Em seguida, ■ acumulador é carregado com ■ valor hexadecimal 7F. Esse número especifica ■ canto inferior esquerdo do teclado. A instrução **in a,254** recebe o padrão de bits correspondente ao estado atual dessa parte do teclado, colocando-o no acumulador.

O bit zero representa ■ estado da tecla <**BREAK**>. Se esta não estiver sendo pressionada, ele valerá 1; ■ estiver, seu valor será igual ■ 0.

Para verificar isso, execute uma rotação, jogando o bit zero — que fica no extremo direito do byte — para o carry bit e checando, depois, ■ sinalizador carry. A instrução ■ promove ■ rotação e ■ ■ retorna ao BASIC quando não ■ um "vai um" — carry igual a zero ou "not carry". Assim, quando pressionar <**BREAK**> e o bit zero pas■ a valer zero, ■ provocará ■ fim da rotina; caso contrário, esta continuará funcionando.

Lembre-se de que um retorno condicional tem o mnemônico *rts* apenas em nosso Assembler. Outros Assemblers utilizam a forma **ret**.

COMPLETE O CÍRCULO

Agora que temos uma saída da rotina, o valor do acumulador é recuperado por **ld a,d**, já que estava temporariamente em D.

A é incrementado para especificar a cor seguinte. A instrução **cmp 7** compara o resultado com 7. Este é o número correspondente ao branco, cor da tela principal. Se o número no acumulador não é sete, **jrnz,loop** volta para enviar cor da próxima banda. Quando este laço tiver contado de 0 a 7, o microprocessador vai à instrução **jr redo**, que retorna ao início do programa, à espera de uma nova varredura da tela.

Você pode estar estranhando o fato de não produzir sons por acidente ao mudar a cor da moldura. A rotina não afeta o bit que controla o alto-falante. O maior número que enviamos pela porta 254 foi 7, quando precisaríamos de, no mínimo, 16 para movimentar o alto-falante.

EFEITOS SONOROS

Veremos agora como criar um efeito sonoro mais complexo, com o comando **out**. A rotina apresentada a seguir produz um som de disparo laser, usando a combinação de um som cuja freqüência é crescente com um som de freqüência decrescente.

```
 10 REM org 64600
 20 REM ld a,(23624)
 30 REM rrca
 40 REM rrca
 50 REM rrca
 60      ld b,0
 70 REM loop push bc
 80      xor $10
 90 REM out 254,a
100      push af
110 REM xor
120 REM sub
130 REM ld b,a
140 REM pop af
150 REM pausea nop
160 REM djnz pausea
170 REM xor $10
180 REM out 254,a
190 REM pop bc
200 REM push bc
210 REM pauseb nop
220 REM djnz pauseb
230 REM pop bc
240 REM djnz loop
250 REM ret
```

As quatro primeiras instruções são exatamente iguais às do outro programa de som. Têm a função de preservar a moldura.

As duas seguintes — que estabelecem os valores iniciais de dois contadores, um no registro e outro colocado na pilha a partir do registro B — também são iguais. O mesmo ocorre com as outras duas, que usam "ou exclusivo" entre os bits quatro dezesseis e mandam o resultado pela porta 254. Desta vez, porém, **xor** é seguido de $10 — 16, em hex. Isto produz o som.

O **push** guarda o conteúdo do acumulador pilha, junto com o registro F. Os registros só podem ser guardados em pares. A instrução **xor** limpa o acumulador e **sub b** subtrai de 0 — conteúdo do acumulador. Na prática, porém, inverte-se o conteúdo de B. Quando é 255, obtemos 1 como resultado da subtração; quando é 1, obtemos como resultado 255.

O **ld b,a** coloca o resultado da subtração de volta em B. O acumulador que estava na pilha é novamente recuperado, retornando ao acumulador.

Temos, então, um laço de pausa cuja duração depende do conteúdo de B — lembre-se de que **djnz** decrementa o registro B e verifica se o resultado é zero. O valor do bit quatro invertido e enviado ao alto-falante.

O **pop** recupera da pilha o valor inicial de B. Como este valor será utilizado mais adiante, é guardado novamente na pilha, depois de ter sido copiado B. Agora ele se encontra, portanto, na pilha e em B.

A próxima pausa depende desse valor inicial de B, que mais uma vez é recuperado da pilha. O mesmo ocorre sempre que se executa uma instrução **djnz**, para restaurar valor de B. Quando o processador sai de um laço **djnz**, o valor de tem que ser zero.

O valor inicial de B é, então, diminuído. Não sendo zero, o processador retorna ao início do laço e executa-o novamente, com um valor menor que o de B. Na passagem 256, quando se torna 0, o processador passa à instrução **ret** retorna ao BASIC.

Você verá que o laço **pausea** é executado 256 vezes na primeira passagem, uma vez segunda uma vez mais a cada nova passagem. O laço **pauseb**, por outro lado, é feito 256 vezes na primeira passagem e uma vez a menos cada nova volta.

ADIÇÃO DE SONS NATURAIS

Os sons que os computadores produ- parecem artificiais porque são muito puros. Instrumentos musicais e outros aparelhos que produzem sons tendem a ser bastante irregulares na maneira de produzi-los. Mas é justamente esta característica de acaso que lhes assegura efei-

FAÇA MÚSICA NO SPECTRUM

O código **out** permite a produção de música em seu micro. Mas esteja preparado: não fácil executar os ajustes necessários para obter freqüência correta.

O uso de uma rotina com dois parâmetros — relacionados à freqüência e duração — simplifica bastante tarefa. E esta rotina existe na ROM. Chamada de rotina BEEP, começa no endereço 03f8. Existe ainda uma rotina adicional chamada BEEPER, que utiliza o valor em HL, para controlar freqüência, e o valor em DE, para controlar a duração.

O valor que se deve usar em HL é dado por 437500/f-301125, onde é a freqüência da nota. Multiplicando freqüência pela duração desejada, obtém-se o valor ser colocado em DE. Com este método, não é preciso observar o limite usual de 10 segundos para duração da nota.

Se você chamar rotina diretamente, deverá colocar na pilha os valores de duração e freqüência que são empregados num comando **SOUND** comum.

tos tão agradáveis. Um instrumento não toca apenas o som fundamental, mas também os harmônicos.

Existe uma maneira de introduzir um elemento de acaso produzidos pelo Spectrum. A RAM, entre 16384 e 32767, fica em circuitos integrados que são geralmente interrompidos por dispositivo que cuida da tela de TV e executa ainda outras tarefas.

Normalmente, essas interrupções são quase imperceptíveis, de tão curtas — em geral duram uns poucos microssegundos. Mas a relação entre sons e o tempo é tão estreita que o ouvido pode captar variações mínimas. Assim, colocando o programa gerador de sons nessa área da memória, eles soarão bem mais naturais — e terão uma duração um pouco maior.

No nosso caso, a experiência não poderá ser feita, pois nosso Assembler ocupa essa área da memória. Contudo, se não estiver usando impressora, tente montar o programa em seu buffer, caso ele caiba ali. O buffer vai de 23296 a 23532 — assim, 23296 como origem. Não preciso proteger esse espaço, pois o buffer está a salvo de ser apagado pelo BASIC.

SUA LIGAÇÃO COM O MUNDO

- CONEXÃO COM O MUNDO
- COMPRAS EM CASA
- TELETEXTO E VIDEOTEXTO
- LIGAÇÃO EM REDE
- TIPOS DE MODEM

■ você deseja explorar plenamente o potencial de seu micro, faça-o comunicar-se com outros computadores, utilizando o modem como elo de ligação entre ele ■ o mundo.

Dos computadores pode-se dizer que, como homens, nenhum deles é uma ilha... Pois até mesmo os mais modestos dentre eles podem ser conectados ao sistema telefônico e, em conseqüência, a alguns dos mais poderosos "cérebros eletrônicos" e bancos de dados da Terra. Ligado desse modo, um micro nos proporciona acesso a quantidades inesgotáveis de informação.

Um computador, um dispositivo de ligação chamado *modem*, um telefone e um software muito simples: eis tudo ■ que ■ preciso para realizar esse milagre da informática.

A comunicação entre computadores ■ dá de três modos básicos: por meio de linhas telefônicas ou de ondas de rádio (ou ambos, em alguns casos); através da co-participação no mesmo sistema de armazenamento de informação (que, na prática, significa co-participação na mesma unidade de disco); ou com a ajuda de um simples cabo que conecte dois aparelhos — neste caso, cada um deles pode compartilhar o processador e a memória do outro.

ALÔ, ALÔ... MUNDO

Dessas opções, a primeira é sem dúvida ■ ■ excitante, pois representa um passo à frente para quem é entusiasta dos computadores pessoais.

Atualmente, já é possível obter, a preços razoáveis, equipamento e software que farão com que ■ seu aparelho seja capaz de conversar com outro em qualquer lugar do globo, usando as mes■ linhas empregadas pelos sistemas de telecomunicações.

O sistema telefônico proporciona acesso a uma imensa faixa de facilidades, como o intercâmbio de software ■ a ampliação dos serviços de notícias ■ informações, que hoje custam muito caro, quando realizados pelos meios convencionais (tele-trabalho).

Com um computador e um modem, você pode consultar listas de compras, examinar ilustrações de produtos para venda, comparar preços, pagar contas instantaneamente, verificar de modo automático como andam as suas finanças, e fazer quase tudo o que quiser, sem sair de casa.

Para trabalhar juntos, uma secretária e seu chefe, por exemplo, não precisam estar na mesma sala, no mesmo edifício ou na mesma cidade. Basta para isso que usem formas de comunicação por computador.

Assim, tendo esboçado uma carta no micro, o chefe pode enviar todo o rascunho, com erros ortográficos e de gramática, à sua secretária, que ■ corrigirá, o formatará de modo a torná-lo apresentável e o enviará.

[illegible]

Diversos sistemas telemétricos, como ■ videotexto e ■ Cirandão, permitem que o usuário estabeleça ■ sua própria "caixa postal". Assim, se ele quiser entrar em contato com outro usuário, pode "escrever" uma carta no processador de textos do micro, e guardá-la na "caixa postal" do destinatário. Este último, por sua vez, poderá copiá-la em seu microcomputador, ■ fim de lê-la mais tarde. Certamente, tanto quem envia como quem recebe a carta deve ter um computador ligado à rede. Um *Correio Eletrônico* desse tipo constitui uma poderosa ferramenta, ■ representa um grande progresso em relação aos boletins.

REDES

Muitas das aplicações mais interessantes para ■ comunicação entre computadores envolvem pequenos aparelhos com acesso a um grande "cérebro eletrônico". Entretanto, outras combinações são possíveis.

Uma vida solitária? Talvez, mas não necessariamente. Também é possível contatar outras pessoas com os mesmos

interessses em computadores. Em alguns casos, independentemente do lugar do mundo em que estejam essas pessoas, o custo do contato com elas pode ser tão pequeno quanto uma simples ligação telefônica local. Você pode chamar aquilo que ■ conhecido por *boletim* (*bulletin boards*, em inglês, comumente traduzido para "quadro de avisos") para ler mensagens enviadas por outras pessoas ■ também deixar suas próprias mensagens. Existem boletins programados para vários propósitos.

Se você decidisse abandonar o conforto de seu lar e aventurar-se ■ sair pelo mundo afora, poderia deixar registrado quase tudo no computador — despesas com táxis e hotéis, dias de férias, entradas para teatro, passagens de avião, ■ muito mais.

Mesmo que surjam problemas nas ligações de computadores (causados sobretudo por falhas no estabelecimento de padrões comuns), quase sempre se dá ■ jeito; muitos desses trabalhos pioneiros são realizados por amadores e estudantes entusiasmados e seus professo■ em universidades ■ outros estabelecimentos de ensino superior.

O crescimento do interesse em comunicações entre computadores e as indicações de que esta será a área de exploração que mais se desenvolverá no futuro refletem-se claramente no fato de que mais e mais boletins computadorizados estão sendo criados em todo o mundo (inclusive no Brasil).

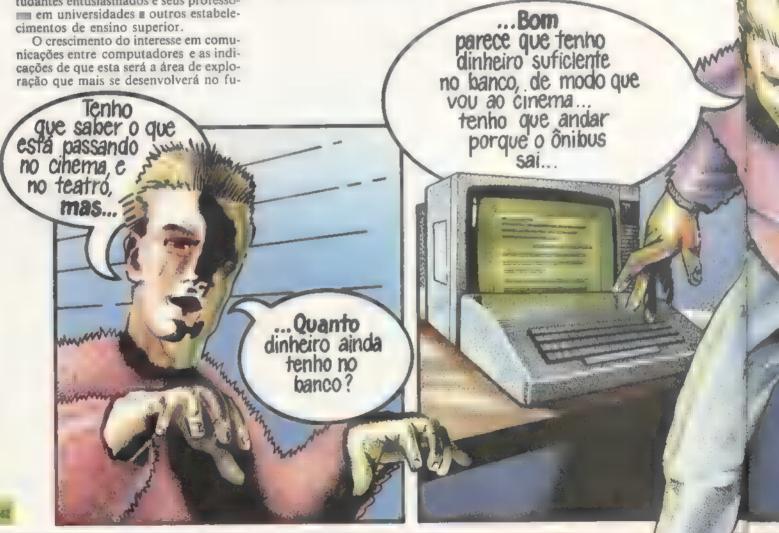
Embora possam ser feitos pelo próprio usuário, os boletins são, em sua maioria, montados por empresas, geralmente especializadas no ramo da eletrônica. Alguns desses "boletins empresariais" reservam um espaço para os usuários comuns, enquanto outros ocupam ■ área do quadro apenas para publicar seus serviços e benefícios. Ultimamente, porém, o setor de maior crescimento tem sido ■ dos boletins criados por amadores e universitários.

COMPRAS EM CASA

Ainda há quem imagine que essa aplicação da comunicação entre computadores pertence ao domínio da ficção científica, e que será necessário esperar muitos anos, até que seja possível realizar, por exemplo, compras por intermédio de um terminal de vídeo em casa. Entretanto, tudo isso já é possível atualmente. Nos países mais desenvolvidos, muitas pessoas utilizam esse sistema, e estão tão familiarizadas com ele, que ■ telecompras já fazem parte de sua rotina diária. No Brasil, o sistema videotexto oferece essa possibilidade em várias capitais.

Nos lugares em que isso já acontece, você pode usar o seu computador para ter acesso instantâneo a informações sobre produtos, ofertas especiais ■ preços em um grande número de lojas. Ou pode pedir serviços ■ pagá-los com ■ simples digitação de algumas teclas. O "trabalho" é feito por um computador central. Ele contém toda a informação e também faz a transferência eletrônica de dinheiro da conta do cliente no banco para a conta da loja.

TELETEXTO E VIDEOTEXTO

Igualmente crescente é o número de serviços de teletexto e videotexto, aos

quais é possível conectar até mesmo pequenos micros domésticos. O serviço de teletexto — que ainda não existe no Brasil — ■ o que sai mais barato para o usuário, pois não é interativo. Desenvolvido ■ Inglaterra, ■ serviço segue dois padrões adotados em vários países: o ORACLE (executado pelas companhias de televisão independentes) e o CEEFAX (executado pela BBC). Com o teletexto, o usuário utiliza um televisor adaptado para receber em casa milhares de páginas de informação (textos ■ figuras, por exemplo), que são mostradas no vídeo. Não é necessário o telefone, pois ■ páginas são enviadas por radiodifusão, ■ intervalo entre uma tela ■ outra da programação normal de TV.

Enviar e carregar programas são duas funções de grande interesse em comunicação de computadores; elas podem ser desempenhadas tanto no videotexto quanto no teletexto. Programas podem ser transmitidos também por TV ou por rádio. Porém, os programas do teletexto têm uma vantagem: eles podem ser carregados diretamente no micro, enquanto os programas obtidos por meio do rádio precisam ■ primeiramente gravados em fita para, só então, serem carregados pelo modo normal.

A única desvantagem do teletexto é que, como ele usa sinais de radiodifusão, ■ comunicação se dá apenas em um sentido. Já o videotexto (que emprega uma linha telefônica para estabelecer comunicação) é bidirecional. Isto significa que a sua máquina pode "conversar" com o computador central do videotexto. Na realidade, a rede de videotexto pode incluir vários computadores regionais interligados, todos oferecendo basicamente o mesmo serviço. Essa multiplicidade de computadores ajuda ■ manter baixo o preço do sistema para o usuário, que já paga taxas altas só para usar ■ telefone. Quanto mais perto ■ computador estiver do assinante, mais barata será a ligação.

Outro serviço público de telecomunicação entre computadores, bem mais diferenciado e sofisticado que o videotexto e ■ teletexto, é constituído pelas redes de microcomputadores, como por exemplo o sistema Cirandão, da empresa estatal Embratel. Esse sistema proporciona aos proprietários de micros a possibilidade de conversarem entre si, empregando uma rede de correio eletrônico ("caixas postais"). Além disso, ele oferece também muitos outros serviços de "boletim computadorizado", teleconferência, consulta ■ bancos de dados, e recepção e transmissão de programas de computador.

Uma desvantagem de sistemas com muitos usuários ■ que utilizam ■ rede telefônica normal, como ■ Cirandão, é ■ sua baixa velocidade de transmissão. Isso acontece porque as redes telefônicas brasileiras não suportam um ritmo muito intenso de transmissão, sem degradação do sinal. Por isso, usuários profissionais (bancos, por exemplo) são obrigados ■ recorrer a redes especiais de transmissão em alta velocidade, como é o caso de outro serviço da Embratel, o Transdata.

O Transdata permite enviar e receber informações em uma faixa maior de velocidade, o que significa também que computadores de diferentes tamanhos e velocidades podem compartilhar os serviços. Estes, porém, são muito mais caros do que os cobrados pelo Cirandão e pelo videotexto.

O acesso a bancos de dados ■ redes de computadores situados em outros

países é possível por intermédio do serviço Interdata da Embratel.

CORREIO ELETRÔNICO

Uma rede pode interligar um computador principal, microcomputadores e qualquer espécie de periféricos. O jornal norte-americano *New York Times*, por exemplo, usa os três tipos de computador em um dos mais sofisticados sistemas de produção de informações. Em sistemas dessa envergadura, muitas pessoas que trabalham em escritório podem usar o seu próprio terminal ligado a um minicomputador, enquanto outro terminal no escritório se comunica com um computador principal.

Desse modo, o trabalho do dia-a-dia, no caso de uma redação de jornal, pode ser preenchido por um simples minicomputador, enquanto o computador central é reservado para o uso da administração da empresa e para os bancos de dados principais.

Por outro lado, os jornalistas que estiverem trabalhando fora da redação podem valer-se de um microcomputador com um modem para ter acesso aos bancos de dados, visando obter informações para suas reportagens. Estas podem ser igualmente enviadas à redação por meio de um computador e um modem.

Em muitos países, é comum o fato de jornalistas ("cobrindo", por exemplo, uma partida de futebol ou uma corrida de Fórmula-1) utilizarem diretamente um teclado e um vídeo, em lugar de uma máquina de escrever. Ao serem apertadas algumas teclas, o resumo e o resultado do jogo são instantaneamente enviados ao computador da redação.

Como os fabricantes de computadores ainda não se ajustaram à necessidade de estabelecer padrões comuns internacionais, a comunicação entre computadores não é, por enquanto, tão fluida como se poderia desejar. Entretanto, os problemas que concorrem para isso estão sendo solucionados e, aos poucos, os computadores vão se tornando compatíveis entre si. Isso tem acontecido principalmente no caso da comunicação entre computadores por intermédio de linhas telefônicas.

SEGURANÇA

O fato de computadores diferentes poderem se comunicar uns com os outros através de linhas telefônicas agrada muitos possuidores de micros, mas assusta universidades e grandes empresas, bancos e agências do governo. O problema é que tais instituições também utilizam esse tipo de comunicação. Isto as torna vulneráveis pois, apesar das medidas de segurança adotadas pelas autoridades para garantir a confidencialidade das informações armazenadas e transmitidas, seus computadores são, vez por outra, "invadidos" por usuários não autorizados.

Esse é o tema do filme *Jogos de Guerra*, que foi objeto de muita controvérsia, especialmente nos Estados Unidos, onde a fraude em sistemas telemétricos tem causado prejuízos de milhões de dólares.

Os crimes eletrônicos são favorecidos pela relativa facilidade com que se processa a comunicação entre computadores. Em vez da violência física, os criminosos utilizam, nesse caso, métodos bem mais sutis e sofisticados. Um dos maiores problemas com que se defrontam as instituições fraudadas tem sido, aliás, saber se estão realmente sendo roubadas. Como muitos serviços são feitos por meio de linhas telefônicas, a investigação do crime eletrônico vai se tornando cada vez mais difícil, particularmente quando o próprio criminoso pode efetuar a mudança dos registros relativos ao crime.

OUTROS MEIOS DE COMUNICAÇÃO

Os telefones não são o único meio de comunicação entre computadores. Ondas de rádio também podem ser usadas, desde que haja dois computadores conectados a equipamentos capazes de transmiti-las e recebê-las.

As fibras ópticas são atualmente o meio pelo qual a comunicação entre computadores se processa de modo mais rápido. Nesse caso, a informação é codificada em impulsos de luz e transmitida por meio de finos fios de vidro. Essa nova tecnologia parece ser mais eficiente do que qualquer outro método conhecido: ela faz com que a informação viaje à velocidade da luz. As comunicações por fibra óptica já passaram da fase experimental, e estão sendo usadas em diversos países, inclusive no Brasil.

Os raios infravermelhos também vêm sendo testados como meio de comunicação entre micros numa mesma sala. A vantagem principal é que se elimina o "cipoal" de fios que acompanha toda grande instalação de computadores.

TORRE DE BABEL

Não importa o meio de comunicação sempre existirá problema de escolhido: compatibilidade. Isso acontece porque um micro só pode se comunicar com outro micro (ou com outro dispositivo periférico) por intermédio da interface correta. Sem a interligação correta de cabos e soquetes e um código que as duas máquinas entendam, a comunicação torna-se impossível.

A linha telefônica é usada para carregar informações de um computador para o outro através de longas distâncias. Entretanto, ela aceita apenas sinais seriais.

Isso não significa que os computadores com interfaces paralelas não sejam capazes de enviar sinais pela linha telefônica. Existem circuitos eletrônicos e programas disponíveis que tornaram possível essa emissão. Tais dispositivos, porém, podem sair mais caros do que o próprio computador.

Um dos problemas com o uso de telefone para comunicação entre computadores é que estes carregam informações em impulsos ou pulsos elétricos discretos e separados. Mas os telefones atuais são projetados para transmitir a voz humana — que é um sinal analógico, contínuo e variável. Assim, o dado de um computador precisa ser convertido em um sinal similar.

MODEMS

O equipamento usado para converter os dados do computador em sinais que possam ser transmitidos através das linhas telefônicas, e vice-versa, é o modem. A palavra é uma abreviação de MOdulador/DEModulador, que é a descrição da função do modem.

Existem dois tipos de modem, o acoplador acústico e o modem de conexão direta. Ambos variam de formas e tamanhos, mas estão geralmente contidos em uma caixa. O acoplador acústico tem duas alças de borracha para acomodar o bocal do aparelho telefônico.

O acoplador acústico é conectado ao micro e transforma seus sinais em tons que podem ser enviados pela linha telefônica. Ele também converte tons de entrada vindos de outro computador em informações digitais que o computador receptor pode entender.

Já o modem de conexão direta codifica (ou modula) o dado do computador diretamente em sinais elétricos e decodifica (ou demodula) a informação de entrada em bits seriais entendidos pelo computador. Esses modems podem transmitir informações a velocidades maiores que os acopladores acústicos e são menos propensos a erros.

COMO FUNCIONA O GERADOR GRÁFICO

- AS POSIÇÕES DE IMPRESSÃO
- MOVIMENTO BLOCO POR BLOCO
- COMO ABRIR CANAIS
- USE TABELAS DE REFERÊNCIA
- COMO MODIFICAR APONTADORES

Incluído ■ programas de animação gráfica para os micros das linhas Spectrum ■ TRS-Color que apresentamos aqui, o gerador gráfico descomplicará a sua vida.

Em artigo anterior da seção *Código de Máquina*, instruímos os usuários do TK-90X ■ do TRS-Color a inserirem dados para ■ montagem de um quadriculado que poderia ser usado na criação de caracteres gráficos (UDG). Nesse artigo, apresentamos um pequeno programa em linguagem de máquina no qual foram inseridos alguns números cujo significado — afirmávamos — não precisava ser compreendido naquele momento.

Mas agora, com os conhecimentos que você já adquiriu a respeito da linguagem de máquina, é possível abordar ■ que significam esses números. Para facilitar ■ compreensão, decodificamos os números que estavam em Assembler, transformando-os em mnemônicos.

Os usuários do MSX, do Apple e do TK-2000 não precisam se preocupar com isso, pois ■ programas para esses micros empregam sprites ■ figuras móveis para criar ■ caracteres gráficos, não exigindo assim rotinas em código de máquina.

S

```
 org 65200
     jr start
mode defb 1
data defb 22
     defb ■
     defb 0
     defb 32
     defb 32
     defb 32
     defb 22
     defb 0
     defb ■
     defb 32
     defb 32
     defb 32
     defb 22
     defb 0
     defb 0
     defb 32
     defb 32
     defb 32
     defb 22
     defb 0
     defb ■
     defb 144
     defb 145
     defb 146
     defb 22
     defb ■
     defb 0
     defb 147
     defb 148
     defb 149
     defb 22
     defb 0
     defb ■
     defb 150
     defb 151
     defb 152
     defb 22
     defb 0
     defb 0
     defb 153
     defb 154
     defb 155
     defb 22
     defb ■
     defb ■
     defb 156
     defb 157
     defb 158
     defb 22
     defb 0
     defb 0
     defb 159
     defb 160
     defb 161
start ld a,(mode)
      cp 1
      ld bc,18
      jr z,print
      jr c,reset
      sla c
      jr print
reset ld c,0
print ld ix,data
      add ix,bc
```

```
ld a,(23689)
ld b,a
ld a,24
sub b
ld (ix+1),a
inc a
ld (ix+7),a
inc a
ld (ix+13),a
ld a,(23688)
ld b,a
ld a,33
sub b
ld (ix+2),a
ld (ix+8),a
ld (ix+14),a
push ix
ld a,2
call $1601
pop de
ld bc,18
call $203C
ret
```

A primeira instrução faz com que o processador passe para o início do programa propriamente dito, saltando sobre a longa lista de dados que a segue. Evidentemente, poderíamos chamar o programa na primeira instrução que aparece após a lista de dados; mas, como veremos a seguir, é mais fácil chamá-lo pelo endereço de origem.

O primeiro byte de dados, situado logo depois do rótulo **mode**, diz ao programa que quadro deve ser usado. O BASIC dá um **POKE** 0, 1 ou 2 nessa posição. O UDG 0 é um quadro vazio que serve para limpar a tela; e os quadros 1 e 2 são caracteres UDG (por exemplo, tanques de guerra, cada um apontando para uma direção). Se nenhum número for especificado, a rotina assumirá o valor 1 para o quadro.

OS DADOS DO UDG

Depois do rótulo **DATA**, aparecem os detalhes de três quadriculados. O Spectrum desenha seus UDG na forma de cadeias, mas primeiro tem que ser informado sobre em que lugar colocá-los. As instruções para isso também estão contidas na cadeia de dados: 22 é o código para o **AT** em BASIC e os dois zeros são bytes vazios que a rotina utilizará mais tarde para definir a posição de impressão.

As três séries de três números 32 formam o quadro vazio de 9 x 9. O número 32 é o código ASCII para o caractere de espaço. Cada série de três forma uma linha do quadro e deve ser precedida pelo seu próprio **AT** e pela posição de impressão. Esse é o quadro 0.

O quadro 1 é feito dos UDG de 144 a 152. Nos dados, eles vêm em grupos de três com seu **AT** e dois bytes livres para a posição de **print**. O quadro 2 é feito dos UDG de 153 a 161.

QUAL UDG?

O **ld a, (mode)** carrega o número inserido, via **POKE**, no byte **mode** do acumulador, e **cp 1** o compara com 1.

O par de registradores BC é então carregado com 18. Na realidade, o número 18 vai para o registrador C, enquanto o B fica vazio. O que interessa durante a rotina principal é o valor do registrador C; mas o registrador vazio B será usado mais tarde, quando chamarmos rotinas da ROM.

Se o número do **mode** carregado no acumulador for 1, **cp 1** ajustará para esse número o indicador de zero. Então, **jr z, print** saltará para a rotina de nome **print**. Lá, o registrador IX é carregado com o endereço do primeiro byte de dados do UDG e somado com o conteúdo do par BC, 18.

Cada quadro contém dezoito bytes de dados: nove para caracteres UDG, três para os **AT** e seis para posições de impressão. Isso faz o registrador IX passar sobre o quadro 0 e ir para o começo do quadro 1. Se o número do **mode** for 0, **cp 1** ligará o indicador de carry. Em seguida, **jr c, reset** saltará para o rótulo **reset** e carregará C com 0. A instrução **add ix, bc** somará então IX ao endereço do rótulo **data**, deixando IX apontando para o começo do quadro vazio.

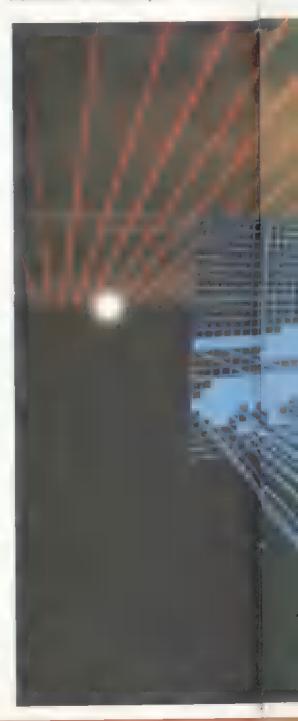
Se o número do **mode** não for 0 nem 1, deverá ser 2. Nesse caso, o processador vai para a instrução **sla c**. Esta significa "rotação aritmética para a esquerda" e age sobre o registrador C. Ela move todos os bits uma casa à esquerda, multiplicando por 2 o conteúdo do registrador.

Assim, ela duplica o 18 (que vai para 36) e salta para o rótulo **print**. Quando BC for somado ao apontador IX, este último será transferido 36 bytes acima na lista de dados, parando na posição do começo do quadro 2.

AS POSIÇÕES DE IMPRESSÃO

A variável de sistema em 23 689 contém a posição de impressão vertical. Ora, a posição de impressão especificada pelo BASIC é a do topo do quadro. Essa variável conta de 1 a 24 de baixo para cima da tela, e não ao contrário, como em BASIC. Portanto, ela deve ser transferida para o registrador B; depois, 24 é carregado em A e o valor de B é subtraído dele.

O apontador IX contém o endereço do primeiro byte do quadro que a rotina irá desenhar. Então **ld (ix + 1), a** carrega a posição vertical de impressão do topo do quadro no próximo byte da lista de dados (o primeiro 0 que estiver vazio).

A é então incrementado para fornecer a posição vertical de impressão da próxima linha de caracteres UDG do quadro. Isso é carregado no oitavo byte da lista por **ld (ix + 7), a**. Depois, A é incrementado novamente para a terceira

linha e isso é carregado no décimo quarto byte.

Essa operação carregou as posições verticais de impressão nos bytes vazios logo após o código do **AT**, 22.

As posições horizontais de impressão provenientes da variável de sistema 23 688 são então carregadas no acumulador. Mais uma vez, essa operação é feita de trás para a frente, se comparada ao que acontece ■ BASIC. Portanto, ■ posição horizontal é carregada em B; A ■ carregado com 33 e B é subtraído de A para fornecer corretamente a posição de impressão.

Como cada linha começa na mesma posição horizontal, isso significa que o ■ valor de A pode ser carregado em outros espaços vazios na lista de dados. As instruções adequadas para fazer isso são **ld (ix + 2),a**, **ld (ix + 8),a** ■ **ld (ix + 14),a**.

O apontador IX é então "empurrado" para a pilha, protegido contra uma eventual alteração causada pela rotina da memória ROM que está prestes ■ ser chamada. A é carregado com 2, ■ rotina de abertura do canal em 1 601 é chamada. O 2 no acumulador define ■ canal a ser usado quando o processador vai para ■ sub-rotina. O canal 1 é a linha de edição na parte inferior da tela; o canal 2 é ■ tela principal; é o 3 ■ impressora.

O apontador IX que foi guardado na pilha é então recuperado no par DE. Depois, BC é carregado com 18, e ■ rotina de impressão de cadeia em 2 032 é chamada. Essa rotina imprime BC caracteres, começando em DE. Portanto, ela imprime os dezoitos caracteres que compõem ■ quadro, ■ começar do endereço base que está no apontador IX.

Feito isto, **ret** retorna o processador para o BASIC.

```
      ORG  32000
      LDX  32700
      LDA  #3
      STA  FCNT
      STA  SCNT
      LDA  #8
      STA  TCNT
      LDA  32250
      BEQ  JUMP
      LDU  #32300
      DECA
      LDB  #72
      MUL
      LEAU D,U
LOOP  LDA  ,U+
      STA  ,X
      LEAX 32,X
      DEC  TCNT
      BNE  LOOP
      LDA  #8
      STA  TCNT
      LEAX -255,X
      DEC  FCNT
      BNE  LOOP
      LDA  #3
      STA  FCNT
      LEAX 253,X
      ■■■  SCNT
      BNE  LOOP
      RTS
JUMP  CLRB
STLP  ■■■  ,X
      LEAX 32,X
      DEC  TCNT
      ■■■  STLP
      LDA  #8
      STA  TCNT
      LEAX -255,X
      DEC  FCNT
      BNE  STLP
      LDA  #3
      STA  FCNT
      LEAX 253,X
      DEC  SCNT
      BNE  STLP
      RTS
FCNT  RMB  1
SCNT  RMB  1
TCNT  ■■■  1
```

O endereço de tela correspondente canto superior esquerdo do quadriculado é armazenado nas posições de memória 32 700 e 32 701 pelo programa BASIC; **LDX** carrega esse apontador de dois bytes no registrador de X dezesseis bits. O número 3 é carregado no acumulador e armazenado dois contadores de nomes **FCNT** **SCNT**. O quadriculado contém 3 x 3 caracteres.

Esses contadores aparecem na lista de dados no fim do programa. Aqui, espaço que o contador vai ocupar reservado por **RMB** (reserva byte de memória). O número 1 que vem a seguir significa que somente um byte de memória deve reservado. Assim, **RMB 50** reservaria 50 bytes de memória.

O terceiro contador, **TCNT**, é ajustado para 8. Depois, conteúdo de 32 250 carregado em A. O número do UDG é armazenado em 32 250; se esse número for zero, **BEQ JUMP** desviará programa para rotina que limpa a tela naquela área. Caso contrário, o processador continuará trabalhando com o UDG propriamente dito.

IMPRIMA GRÁFICOS

O lugar da memória onde os UDG ficam guardados começa na posição 32 300; esse número armazenado U para facilitar os cálculos que definem onde se inicia UDG. O primeiro caractere começa no princípio dessa área; portanto, seu equivalente 0. Note que, no acumulador, o número de caractere pedido é decrementado.

#72 carrega o registrador B com 72, e **MUL**tiplica conteúdos de A B, guardando o resultado D. Os acumuladores A e são registradores de oito bits; D é um registrador de dezes bits; por isso, ele não é sobrecarregado com o resultado. Existem 72 bytes cada gráfico. Cada caractere contém oito bytes há nove caracteres para cada quadriculado (9 = 72). Essa operação calcula fator necessário para se achar endereço inicial do UDG apropriado.

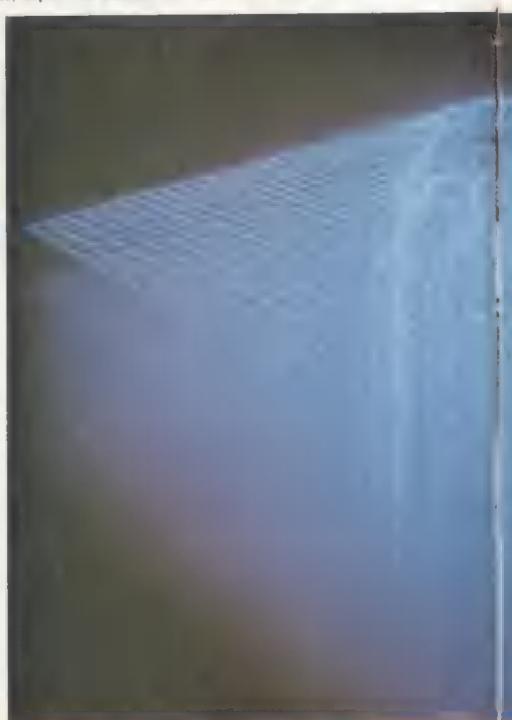
A instrução **LEAU D,U** carrega U com o valor de D mais U. Em outras palavras, ela vai contando ao longo da lista até que seja apontado o começo do caractere pedido.

LDA ,U + carrega acumulador com o conteúdo daquela posição de memória; depois, incrementa o registrador U e prepara-se para trabalhar com o próximo. **STA ,X** armazena byte do caractere obtido da lista de gráficos na posição de memória apontada por X. Lembremos que registrador X contém a posição do canto superior esquerdo do bloco tela gráfica.

LEAX 32,X 32 ao registrador X; isso move o apontador para próxi linha (há 32 posições em cada linha). O contador **TCNT** é então decrementado; ele não for igual zero, a instrução retornará endereço **loop** pa trabalhar com próximo byte do caractere.

O valor inicial de **TCNT** 8; por isso, **loop** deve executado oito vezes. Precisamos de oito bytes, dispostos uns sobre outros, para formar um caractere. Podemos chamar **TCNT** de contador de bytes de caractere.

ADICIONE

LDA #8 e **STA TCNT** retornam valor para o contador **TCNT**, que já está pronto para trabalhar com o pró caractere.

Em seguida, ■ instrução **LEAX -255,X** move ■ apontador X para o início do caractere à direita.

Devemos lembrar ainda que o registrador X contém ■ apontador de tela, e foi oito vezes incrementado de 32 para imprimir totalmente ■ caractere (32 ■ 8 = 256).

O contador **FTNT** é decrementado de modo a percorrer toda a matriz formada por nove caracteres (ou seja, três de cada lado). Quando ■ resultado do decrementado for diferente de 0 (zero), o processador voltará para **loop** e trabalhará com ■ primeiro byte do caractere seguinte.

Se o resultado for 0 (zero), o processador continuará pelo programa.

■ PERCORRER O QUADRICULADO

O contador **FTNT** é então reajustado para 3 ■ o apontador X, adiantado 253 posições. Lembre que, anteriormente, ele havia sido adiantado 32 posições, o que fez com que o apontador fosse para ■ posição de tela diretamente abaixo do caractere que acabara de ser impresso. Em seguida, ele foi atrasado em 255, o que o deixou ■ posição de tela imediatamente à direita do topo do último caractere.

Se desenharmos um pequeno quadriculado mostrando os caracteres e observarmos as posições relativas das locações de tela, veremos que ■ registrador X foi adiantado em 32 posições. Ele aponta agora ■ posição de tela diretamente abaixo do último caractere impresso ■ está duas locações à direita do lugar em que deveria estar ■ primeiro byte do primeiro caractere na segunda fileira.

Portanto, devemos subtrair 2 do apontador X, a fim de movê-lo para ■ posição correta.

Mas, como ■ programa já havia subtraído 255, basta somar 253 que iremos para ■ lugar certo.

O contador **SCNT** é decrementado; ele conta as três linhas de caracteres que formam o gráfico inteiro. Quando o resultado do decremento for diferente de 0 (zero), o **BNE** levará o processador de volta para trabalhar com ■ linha seguinte.

Todavia, se o resultado for 0 (zero), isso significa que ■ gráfico foi impresso totalmente; nesse caso, **RTS** levará o processador de volta para o programa BASIC.

Siga atentamente as instruções, que o processo acabará ■ tornando claro ■ compreensível.

LIMPE A TELA

Quando o número do bloco a ser impresso na tela for igual a zero, a rotina simplesmente limpará a área da tela que está sendo acessada. O modo pelo qual se limpa a tela é praticamente o mesmo pelo qual ■ imprime nela. A diferença entre eles é que a rotina não precisa consultar a lista de caracteres, ■ somente imprimir o código 0, ou seja, nada.

CLRB ajusta ■ registrador B para 0. O 0 é então armazenado nas posições de tela apontadas pelo registrador X. Se você quiser se movimentar pelo quadriculado, deve atualizar repetidamente o registrador X, da mesma maneira que na rotina de impressão; só que, desta vez, ■ 0 em ■ é armazenado em cada posição. Uma vez mais, o ■ retornará o processador ■ BASIC para rodar o resto do programa.

CONJUNTOS DE BLOCOS GRÁFICOS (3)

Não se trata de mais uma fábula de Esopo ou de La Fontaine: aqui, o macaco e a cobra se reúnem na floresta para ensinar os segredos do trabalho com blocos gráficos.

Nos artigos anteriores, aprendemos a criar um grande número de blocos gráficos e começamos a trabalhar com grupos de caracteres para fazer um cenário de floresta.

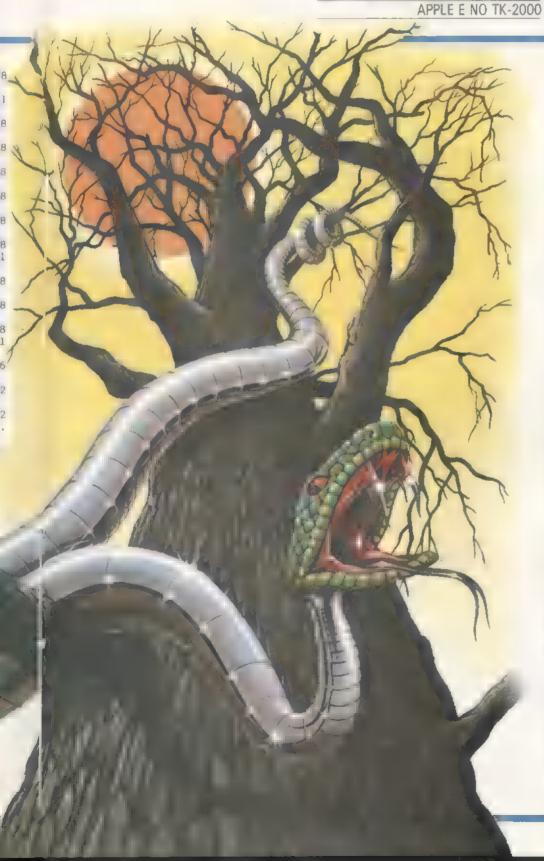
O programa que faz o desenho no último artigo está dividido em partes (com exceção do Apple e do TK-2000, para os quais publicamos apenas o banco de blocos a ser digitado por intermédio do monitor). Cada seção cuida de um segmento (ou figura) do desenho — crocodilo, elefante, árvores, e assim por diante. A seção deste artigo acrescenta uma cobra e um macaco, além de concluir o fundo do quadro. Tanto no caso do Apple como no do TK-2000, apresentamos o programa completo.

Carregue agora a primeira parte do programa para poder digitar a continuação. Os usuários do Apple e do TK-2000 devem antes carregar o banco de blocos gravado no artigo anterior por intermédio do monitor.

S

```
140 POKE 23676,254
150 FOR n=USR "a" TO USR "r"+7
: READ a: POKE n,a: NEXT n
160 POKE 23676,253
170 FOR n=USR "a" TO USR "j"+7
: READ a: POKE n,a: NEXT n
190 POKE 23676,252
200 FOR n=USR "a" TO USR "u"+7
: READ a: POKE n,a: NEXT n
210 POKE 23676,251
220 FOR n=USR "a" TO USR "u"+7
: READ a: POKE n,a: NEXT n
230 POKE 23676,250
240 FOR n=USR "a" TO USR "o"+7
: READ a: POKE n,a: NEXT n
510 INK 1
520 POKE 23676,254
530 LET z=144: FOR n=2 TO 7:
FOR m=16 TO 18: PRINT AT m,n;
CHR$ z: LET z=z+1: NEXT m:
NEXT n
540 POKE 23676,253
550 LET z=144: FOR n=8 TO 12:
FOR m=17 TO 18: PRINT AT m,n;
CHR$ z: LET z=z+1: NEXT m:
NEXT n
610 INK 0
620 POKE 23676,252
630 PRINT AT 0,30;CHR$ 144;AT
1,24;CHR$ 145;CHR$ 146;AT 1,28
;CHR$ 147;CHR$ 148
640 PRINT AT 2,23;CHR$ 149;
CHR$ 150;CHR$ 151;CHR$ 32;CHR$
152;CHR$ 153
650 PRINT AT 3,23;CHR$ 154;
CHR$ 155;CHR$ 32;CHR$ 32;CHR$
156;CHR$ 157;CHR$ 158
660 PRINT AT 4,24;CHR$ 159;
CHR$ 160;CHR$ 32;CHR$ 161;CHR$
162;CHR$ 163;CHR$ 164
670 POKE 23676,251
680 PRINT AT 5,24;: FOR n=144
TO 151: PRINT CHR$ n;: NEXT n
690 PRINT AT 6,24;CHR$ 152;
CHR$ 153;CHR$ 153;CHR$ 153;
CHR$ 153;CHR$ 154;CHR$ 155;
CHR$ 156
700 PRINT AT 7,23;CHR$ 157;
CHR$ 158;CHR$ 159;CHR$ 32;CHR$
32;CHR$ 160;CHR$ 161;CHR$ 162;
AT 8,22;CHR$ 163;CHR$ 164;
710 POKE 23676,250
720 PRINT CHR$ 144;CHR$ 145;
CHR$ 32;CHR$ 32;CHR$ 146;CHR$
147;AT 9,24;CHR$ 148;CHR$ 149;
AT 10,24;CHR$ 150
855 INK 6
860 FOR n=0 TO 2*PI STEP .05:
PLOT 70,150: DRAW SIN n*12,
COS n*12: NEXT n
870 FOR n=0 TO 2*PI STEP PI/4:
PLOT 70,150: DRAW COS n*20,SIN
n*20: NEXT n
970 INK 0
```

MSX

```
10 FOR I=0 TO 983
150 CIRCLE (50,30),25,11:PAINT
(50,30),11
160 DRAW "BM50,30C11NU30NE60NR8
4NF60ND84NG60NL84NH30"
200 FOR I=1 TO 131
3090 DATA 641,18,210,673,19,210
,705,20,210,642,21,210
3100 DATA 674,22,210,706,23,210
,643,24,210,675,25,210
3110 DATA 707,26,210,644,27,210
,676,28,210,708,29,210
3115 REM APAGAR
3120 DATA 645,30,210,677,31,210
,709,32,210,646,33,210
3130 DATA 678,34,210,710,35,210
,679,36,210
3135 REM APAGAR
3140 DATA 711,37,210,680,38,210
,712,39,210,681,40,210
3150 DATA 713,41,210,682,42,210
,714,43,210,683,44,210,715,45,2
10
3160 DATA 23,46,148,49,47,148,5
0,48,148,53,49,148
3170 DATA 54,50,148,80,51,148,8
1,52,148,82,53,148
3180 DATA 84,54,148,85,55,148,1
12,56,148,113,57,148
3190 DATA 116,58,148,117,59,148
,118,60,148,145,61,148
3200 DATA 146,62,148,148,63,148
,149,64,148,150,65,148
3210 DATA 151,66,148,177,67,148
,178,68,148,179,69,148
3220 DATA 180,70,148,181,71,148
,182,72,148,183,73,148
3230 DATA 184,74,148,209,75,148
,210,76,148,211,76,148
3240 DATA 212,76,148,213,76,148
,214,77,148,215,78,148,216,79,1
48
3250 DATA 240,80,148,241,81,148
,242,82,148,245,83,148
3260 DATA 246,84,148,247,85,148
,271,86,148,272,87,148
3270 DATA 273,88,148,274,89,148
,277,90,148,278,91,148,305,92,1
48,306,93,148,337,94,148
3280 DATA 500,95,226,501,96,226
,502,97,226,503,98,226
3290 DATA 532,99,226,533,100,22
6,534,101,226,535,102,226
3300 DATA 536,103,226,565,104,2
26,566,105,226,567,106,226,568,
107,226
```

```
3310 DATA -32,-4,36,75,80,147
3320 DATA 342,108,36,343,109,36
,374,110,36,375,111,36
3330 DATA 406,112,36,407,113,36
,439,114,96,471,114,96
3340 DATA -44,-20,-15,60,84,96,
157,163
3350 DATA 327,115,36,328,116,36
,329,117,36,330,118,36
3360 DATA 360,119,96,361,120,96
,392,121,96,424,121,96
3370 DATA 456,122,96
```

```
1400 REM COBRA
1410 DATA 0,0,0,0,0,0,3,14,31,3
1,63,67,65,254,254,225,71,63,0,
0,0,0,0,0,0
1420 DATA 0,0,0,0,255,12,28,28,
28,254,143,1,0,4,248,208,224
1430 DATA 0,0,0,0,0,0,0,0,0,0,0
,128,240,56
1440 DATA 52,99,225,225,225,96,
47,31,14,6,2,1,0,0,0,0,0,0,0,0,
3,5,63,127
1450 DATA 255,225,240,248,252,2
55,255,126,60,28,124,247,247,11
5,31,1,0,0,0,0
1460 DATA 255,191,159,143,199,1
94,127,64,128,128,64,63
1470 DATA 48,112,248,249,253,25
5,189,255,0,0,0,0,192,252,190,6
3,63,24,240
1480 DATA 9,5,7,5,249,67,243,25
1,255,252,238,239,255
1490 DATA 192,224,208,223,224,1
29,193,227,243,247,255,248,240,
224,192
1500 DATA 0,0,0,0,255,129,195,1
99,199
1510 DATA 239,239,255,0,0,0,0,0
,0,0,0,128,248,31,140,158,191,1
91,255,0,0,0,0
1520 DATA 0,0,0,1,15,62,206,30,
61,121,243,247,247,63,15,3,0
1530 DATA 30,127,191,95,93,61,2
50,240,194,252,194,192,192,224,
224,240
1540      MACACO
1550 DATA 1,2,4,8,16,32,64,128,
0,0,0,3,31,63,127,252,0,0,0,224
,248,252,252
1560 DATA 62,3,15,15,15,15,14,1
4,12
1570 DATA 3,142,240,192,128,0,0
,0,1,1,3,3,3,3,3,3,248,240,224,
224,192
1580 DATA 192,192,192,30,14,110
,76,56,0,0,0,0,0,0,0,0,0,0
```

```
1590 DATA 1,28,28,56,56,112,112
,224,224
1600 DATA 3,3,1,1,1,0,0,0,192,2
24,224,240,248,248,252,126,3,7,
15,31,63,62
1610 DATA 126,252,192,192,128,0
,0,0,3,7,0,0,0,0,0,0,128,223
1620 DATA 127,63,31,31,15,7,7,3
,0,128,128,192,192,192,224,240,
252,126,126,127
1630 DATA 63,63,31,31,15,15,7,3
,3,131,131,131
1640 DATA 255,255,231,231,255,2
52,172,183,128,192,224,224,224,
224,224,224
1650 DATA 1,1,0,0,0,1,3,7,248,2
55,255,255,255,255,255,255,0
1660 DATA 255,255,255,255,255,2
55,255
1670 DATA 15,255,255,255,255,25
5,255,255,199,255,255,255,255,2
55,255,255
1680 DATA 223,239,255,255,252,1
28,0,128,224,240,248,248,240,96
,0,0,0,0,200
1690 DATA 232,248,248,120,112
1700 DATA 7,15,15,31,31,31,63,6
3,255,255,255,255,255,255,255,2
55,192,224,224
1710 DATA 224,224,224,192,192,0
,0,1,1,3,7,14,60,240,224,192,12
8,128,0,0,0
1720 DATA 0,0,0,0,0,3,15,31,63,
63,63,63,126,254,252,240,191,19
1,63,63,63
1730 DATA 63,63,63,15,7,7,7,7,7
,7,7,192,131,135,159,191,255,25
4,252
1740 DATA 252,248,240,224,192,1
28,0,0,0,1,3,7,6,6,4,0,127,254,
240,224
1750 DATA 64,64,0,0,192,0,0,0,0
,1,1,3,63,31,63,127,254,252,248
,240
1760 DATA 7,3,0,0,0,0,0,0,240,1
92,0,0,0,0,0,0,7,7,15,31,31,31,
23,23,224
1770 DATA 192,128,0,0,0,0,0,7,6
,4,0,0,0,0,0
```

T

```
20 DIM C(59),M(156),S(48),E(17)
,T1(1),T2(7),F1(7),F2(7)
90 PCLS 3
100 DRAW "BM62,1C2DG6LG2LGLUHLG
LD6GDGDGDG7DG2DRD3RD2FDFDL12H2L
ULU3HUH4UH2ULU5EUE4R3F3D2GLHURB
D2R2E2U3HUHLHL4GL2G4DGD7FD2FDFD
F2D"
110 DRAW "F2DFD2G3DGDGD2GD5G2L2
GLG2LGLG2D3EUER2ND2R2R2ER2ERERE
UEUEUERD9FG4DG2DG2D2ND2R2D5E2U5
E8U9E3R17F3D6FR3ERERE2RE5UE3UE3
UE"
120 DRAW "U2NU2LH2LD3FDGDGDG4LG
LGLG2LG2HU2EUEU4H3E2R4ERFRFEURU
H2U6H3L4GH2L2G2DF2D4G2L2H2U2HUH
UHUEUEUE2UEUEUEUEU2EUEREUL4D5L3
D6L2D3L2D3"
130 DRAW "BM24,16U3NL6DL8D2L2D2
L2D9BM70,43D4G8BM3,64E3BM+10,10
D3"
140 PAINT (50,50),2
150 DRAW "BM56,34C1RDBF3DBL7DF2
BE2BU2"
160 GET(0,0)-(76,80),M,G
170 PCLS
180 DRAW "BM13,11C4G3L8H2U2EUE2
ERERER8FR2F3RE3RERER11FR3F2RF4R
12FR3FR4ERER2ERERER3FD4GBU3GDG2
LGFNR3GD2FDFL4"
190 [illegible] "H2LHLHLHL18G4LGL16HL3
HLH9LHL2HL2G3DFR3BM23,6D3FDF5RF
2R9ER2ER7BM52,8LGL3G2L11H2UE2R7
F4"
200 PAINT (60,10),3,4:PAINT (13
,5),3,4
210 DRAW "BM83,8C1DBL4BG5H3G4H5
G5H4G5H3G4H4G3H8E4F2E2F2E2F2E2F
4BM7,13LH3E6F4E4FDFD2F9R2FR15E"
220 GET(0,0)-(88,21),S,G
370 CIRCLE(50,30),25,2:PAINT (5
0,30),2:DRAW "BM50,30C2NU30NE60
NR84NF60ND84NG60NL84NH30"
430 PUT (10,166)-(98,187),S,PSE
T
440 PUT (140,0)-(216,80),M,PSET
```

Essas adições encaixam direitinho nos programas do artigo anterior — exceto no MSX, em que várias linhas devem ser substituídas e duas delas apagadas. O que fizemos foi criar mais blocos gráficos. Novamente as linhas **DATA** de 1 400 a 1 770 são comuns ao Spectrum e ao MSX. O programa do Spectrum começa com novas linhas que colocarão na memória os dados adicionais, usando o comando **POKE**.

A listagem do MSX teve o comprimento do laço da linha 10 mudado, a fim de que [illegible] novos dados fossem colocados na parte alta da memória. As linhas 150 e 160 desenham o sol. O laço

da linha 200 também teve seu comprimento modificado para que ■■ posições ■ cores dos novos blocos pudessem ser lidas. Essas novas posições ■ cores se encontram nas linhas **DATA** a partir de 3 090. Observe que parte das linhas do antigo programa teve que ser apagada, pois as posições dos blocos no banco foram modificadas.

O TRS-Color usa **DRAW** para desenhar os novos personagens e **GET** para armazenar os padrões em matrizes.

Cada programa coloca os blocos nas posições adequadas no cenário.

O primeiro passo consiste em carregar o banco de blocos fornecido, por meio do monitor. Se você gravou o seu, entre ■ monitor com:

```
CALL -151
```

posicione a fita e digite:

```
4000.5000 L
```

Ou recorra ao gerador de blocos, se o seu Apple usa disquete. Eis o programa completo:

```
10  HGR :E = 16384:T = 8192
20  HCOLOR= 1: FOR I = 0 TO 176
: HPLOT I,159 - 4 ■  SQR (176 -
 I): NEXT
30  FOR I = 120 TO 279: HPLOT ■
,130 - ■ ■  SQR (I - 120) + (I
- 200) * (I > 200) / 10: NEXT
40  HCOLOR= 5: FOR I = 0 TO 7 S
TEP .05:F = ( INT ( RND (1) * 1
0) - 5)
50  HPLOT 50 + (30 + 20 * F) *
 COS (I),50 + (26 + 20 * F) ■
SIN (I) TO 50 + (30 + 20 * F) *
  COS (I + 3.14),50 + (26 + 20
■ F) ■  SIN (I + 3.14): NEXT I
60  READ NF: FOR M - 1 TO NF: R
EAD L,NL
70  FOR J = 1 TO NL: READ BI,C,
NB: FOR K = 1 TO NB
80 Y = L + J - 1:X = C + K - 1:
N = BI + K - 1
90  FOR I = 0 TO 7
100  POKE T + (Y - ■ ■ (Y > 7)
- 8 ■ (Y > 15)) * 128 + 40 ■ (Y
 > 7) + 40 * (Y > 15) + X + 102
■ * I, PEEK (E + N * 8 + I)
110  NEXT I,K,J,M
120  END
150  POKE T + (Y - 8 ■ (Y > 7)
- ■ * (Y > 15)) * 128 + 40 ■ (Y
 > 7) + 40 * (Y > 15) + X + 102
4 * I, PEEK (E + N * 8 + I)
1000  DATA      7,17,3,1,25,14,4
1,25,14,81,25,14
1010  DATA  17,3,160,4,7,200,4,
13,240,4,13
1040  DATA      3,7,30 ,30,9,70,
30,9,110,30,9,154,34,1,194,34,1
,234,34,1,273,33,3
1050  DATA      5,7,30,32,7,70,3
2,7,110,32,7,154,36,1,194,36,1,
234,36,1,273,35,3
1060  DATA      5,7,30,28,9,70,2
8,9,110,28,9,154,32,1,194,32,1,
234,32,1,273,31,3
1070  DATA       7,7,30,30,9,70,
30,9,110,30,9,154,34,1,194,34,1
,234,34,1,273,33,3
1080  DATA        0,10,25,25,1,5
8,18,7,97,17,7,137,17,7,178,18,
8
1090  DATA         219,19,9,258,
18,9,297,17,9,15,17,8,57,19,1
```

No TK-2000 entramos no monitor por meio de **LM** (e o banco de blocos deve ser carregado em outra posição):

```
A000.B0000 L "nome"
```

A floresta no TRS-Color.

Na linha 10, o valor de T deve ser mudado para 40 960. Os usuários desse micro também podem mudar as cores nas linhas 20 a 40. Para modificar ■■ cores dos animais é preciso alterar de uma unidade a coluna na qual ■ impressa a figura. (Azul e verde sairão invertidos no TK-2000.)

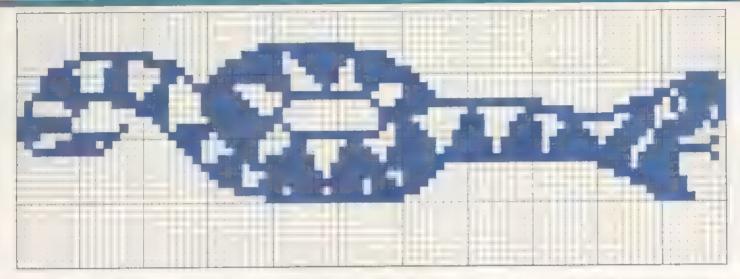
COMO FUNCIONA

A linha 10 liga ■ tela gráfica e armazena em E e T os endereços iniciais das duas páginas de vídeo.

As linhas 20 e 30 desenham as colinas, e as linhas 40 e 50, o sol. Como não há comandos específicos para traçar elipses no Apple, tivemos que lançar mão de fórmulas matemáticas.

As linhas 60 a 120 usam quatro laços **FOR ... NEXT** para colocar os blocos em posição correta na tela. Os dados necessários para tanto estão nas linhas **DATA** de 1 000 em diante.

O método posto em prática é ■ seguinte: inicialmente, a linha 60 usa **READ** para descobrir o número de figuras (**NF**) que serão desenhadas. O laço **FOR ... NEXT** que se inicia nessa linha cuida para que o processo de desenhar uma figura seja repetido **NF** vezes. A cada figura desenhada, são obtidos ■ seguir, também com **READ**, a linha (**L**, entre 0 ■ 39) em que vai ser colocado ■ canto superior esquerdo da figura e o número de linhas da figura (**NL**).

O primeiro **FOR ... NEXT** da linha 70 cuida de desenhar **NL** linhas. O programa tem que saber qual o número do bloco inicial de cada linha da figura *no banco de memória* (**BI**), em que coluna da tela ela começa (**C**) ■ quantos blocos ela tem (**NB**). O segundo **■■■** ... **NEXT** da linha 70 cuida de desenhar **■■** blocos.

A linha 80 se dedica a calcular a posição exata da tela (**X,Y**) e ■ número do bloco no banco (N), para que o laço entre as linhas 90 e 110 possa desenhar um novo bloco.

Esse tipo de organização das linhas **DATA** só é possível devido à disposição dos blocos no banco — carregue ■ gerador e observe o banco. As linhas 1 000 ■ 1 080 contêm os dados do crocodilo, da cobra, das quatro árvores e do macaco.

A linha 1 010, por exemplo, encerra os dados da cobra. Ela nos informa que esta deve ser desenhada com seu canto superior esquerdo na linha 17 e tem três linhas de blocos. O primeiro bloco da primeira linha ocupa a posição 160 no banco e será colocado na coluna 4. Essa primeira linha tem sete blocos. Com um programa assim você poderá fazer seus próprios cenários.

O CENÁRIO COMPLETO

Ao ser executado o programa, a tela se enche de blocos gráficos ■ o cenário parece completo. Uma cobra, um macaco e um sol radiante foram acrescentados pelas novas linhas. (Menos no Apple, em que todas as linhas são novas.) Poderíamos ter incorporado mais elefantes, árvores ou crocodilos à cena, mas, seja como for, dispomos agora de todo um conjunto de novos blocos para usar em nossos desenhos.

O SOL EM ALTA RESOLUÇÃO

Note que, além dos blocos, foram usados comandos gráficos (não há nenhum motivo para não se fazer a combinação das duas técnicas, já que isso dá excelentes resultados).

O Spectrum e o Apple constroem seu sol desenhando vários raios bem próximos, enquanto o MSX ■ o TRS-Color usam o comando **PAINT**.

Podemos ainda alterar o desenho empregando outros comandos gráficos, desenhando mais colinas ou traçando retas que façam ■ chão parecer irregular ou rachado.

Já comentamos ■ necessidade de usar matemática tanto no Apple como no TK-2000 devido à falta de recursos gráficos do BASIC.

Quando se trabalha com blocos gráficos, as variações possíveis são quase infinitas. Com eles, pode-se não só modificar ■ número de árvores, cobras e macacos, como também formar novas figuras. Um exemplo óbvio em nosso caso seria mudar a cor da copa das árvores (para branco ou cinza) e desenhar nuvens ou arbustos.

Devemos, contudo, prestar atenção ■■ cores. Se quisermos, por exemplo, ■■ as copas como arbustos, teremos que colocá-las no topo das colinas já que ambas são verdes. No caso do MSX, podemos mudar o tom de verde; ■■ Apple não haverá problemas, já que ■ colina não foi pintada.

ANIMAÇÃO

As vantagens de construir as figuras com blocos gráficos não terminam aí: podemos também convertê-las em figuras móveis, fazendo certas modificações. Para animar os bichos, poderíamos definir um ou dois blocos extras — construindo um novo corpo para o elefante, por exemplo, ou uma cauda em posição diferente para a cobra, ou mesmo dando ao macaco uma banana para comer.

No MSX, podemos usar sprites para animar as figuras. Já no Apple e no TK-2000 encontraremos algumas dificuldades, pois os blocos gráficos são um tanto lentos. A alternativa é lançar mão do comando **DRAW**, com a ressalva de que, com sua utilização, é muito difícil preservar ao mesmo tempo a forma e a cor de uma figura. As linhas **DATA** necessárias para um **DRAW** são completamente diferentes das linhas que os blocos gráficos empregam.

JUNTE BLOCOS PARA FORMAR FIGURAS

Se quiser empregar blocos gráficos para animar os desenhos, as figuras destas páginas lhe mostrarão como os vários caracteres compõem a cobra e o macaco (o artigo anterior mostra um desenho semelhante para o crocodilo). Esses desenhos não correspondem às figuras do Apple, mas podem ser usados pa■ criar elementos móveis (**DRAW**); para isso, é preciso apelar para o editor de figuras desse micro.

Enquanto o crocodilo, a cobra e o macaco parecem ter seu uso restrito aos desenhos de selvas e zoológicos, ■■ árvores, nuvens e arbustos podem aparecer em outros desenhos.

GRAVAÇÃO DE BLOCOS

Os programas considerados até aqui empregam um grande número de blocos gráficos a título de ilustração, ■ para fornecer blocos potencialmente úteis em outros desenhos.

Todavia, é possível fazer desenhos muito interessantes usando uma quantidade bem menor de blocos. O truque para isso consiste em utilizar diversas ve-

zes os mesmos blocos. Desse modo, podemos, por exemplo, desenhar uma manada inteira de elefantes, ou colocar um muro em nosso cenário, preenchendo uma grande extensão da tela com apenas dois tipos de blocos gráficos.

Entretanto, como a produção de blocos não apresenta problemas, a grande vantagem em utilizar poucos blocos está em economizar tempo (no trabalho de criação e digitação) e espaço de memória (que as longas linhas **DATA** deixam de ocupar. O Spectrum, em particular, gasta muita memória para guardar ■■ linhas **DATA**).

Ao empregarmos linhas **DATA** para armazenar os padrões dos blocos, estaremos ocupando o dobro do espaço necessário, já que esses padrões serão transferidos para uma outra área da memória assim que rodarmos ■ programa. Em compensação, podemos economizar memória, se gravarmos os bancos de blocos diretamente em fita cassete, apagando a seguir as linhas **DATA**.

Para gravar os padrões do banco de blocos, use:

```
SAVE "nome" CODE x,y
```

onde x é o endereço inicial do segmento de memória que queremos gravar, e y é o seu comprimento.

Devemos, portanto, saber ■ endereço inicial do banco de blocos, tanto pa■ colocar nele os valores das linhas **DATA**, como para mudar o apontador de caracteres UDG.

O comprimento do bloco é de fácil dedução: normalmente, ele é o número de blocos multiplicado por 8. O programa visto linhas atrás ■■ oito bancos para guardar todos os blocos necessários ao desenho da selva. Cada um desses bancos começa 256 bytes acima do banco anterior (ao contrário do número mínimo de 168 bytes), a fim de permitir que o apontador seja modificado com apenas um **POKE**, em vez de dois. Isso também significa que, se quisermos gravar essa porção de memória, devemos informar que ela tem 256 x 8 bytes d■ comprimento.

Grave os blocos da floresta com:

```
SAVE "floresta" CODE 63448,2048
```

Para carregar os blocos de volta, digite:

```
LOAD "" CODE
```

Para colocá-los em outro lugar da memória, escreva ■ novo endereço inicial após **CODE**.

O MSX grava o banco de blocos empregando o seguinte comando:

```
BSAVE "CAS:nome" x,y
```

onde x é o endereço inicial e y, o comprimento da parte da memória que queremos gravar.

No caso do programa acima, use:

```
BSAVE "CAS:SELVA" &HD100,&H1000
```

Isso grava apenas o padrão dos blocos. Lembramos que ficam faltando os bytes de cor. (Os blocos não serão visíveis pelo gerador.)

Para carregar ■■ bytes da fita, use:

```
BLOAD "CAS:"
```

Para gravar os blocos guardados na página 2 de vídeo, devemos entrar no monitor com:

```
CALL - 151
```

Posicione ■ fita e digite:

```
4000.5000 W
```

Para carregar o banco, ainda no modo monitor:

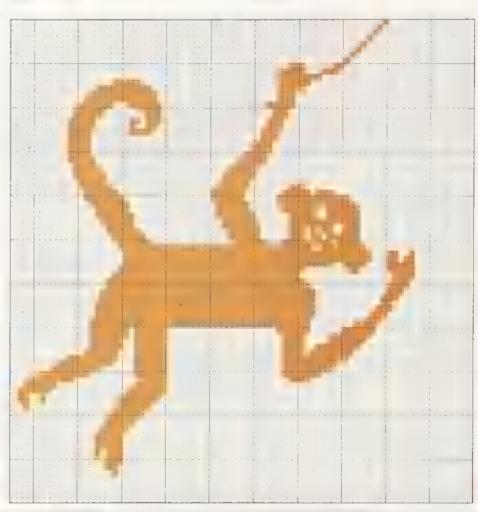
```
4000.5000 ■
```

No TK-2000 o processo se assemelha ao usado no Apple, só que devemos entrar no monitor com **LM**, ■ digitar:

```
A000.B000 W "floresta"

A000.B000 R ""
```

(para gravar e ler ■ fita, respectivamente).

Esse computador usa comandos **DRAW** em vez de linhas **DATA**, de forma que não há vantagem em gravar os blocos.

UM EDITOR DE TEXTOS (1)

Datilografe textos ■■ gastar dinheiro com programas sofisticados ■ sem quebrar ■ cabeça com processadores complicados de palavras, executando o programa deste artigo.

O processador de textos é uma das mais úteis aplicações para qualquer computador pessoal. Em certos casos, quando se quer melhorar os resultados, pode-se comprar um programa sofisticado em linguagem de máquina. Programas como esse, porém, são caros e o uso do editor nem sempre é simples.

A parte do programa coberta por este artigo constitui um padrão para um editor de textos simples, que está estruturado de forma a ser útil até mesmo àqueles que nunca viram um programa desse tipo. Com ele é possível criar memorandos, cartas ou quaisquer outros textos ■ serem enviados para ■ impressora. O texto pode ser gravado em disco ou em fita e carregado para uma posterior utilização.

Devido ■■ tamanho do programa completo — pelo menos 7,5k — a listagem foi dividida em três unidades, que não funcionarão adequadamente até que ■ última parte seja digitada. Apesar disso, será possível criar arquivos de texto no fim da segunda parte. A terceira parte é formada pela rotina para ■ impressora, que possibilita o envio de dados para impressão. Ordenação e rotina de busca estarão disponíveis.

Vejamos como o programa funciona, antes que sua primeira parte seja digitada. Ele está destinado a várias aplicações domésticas; mas, como em qualquer outro programa desse tipo, seus menus e telas de apresentação podem ser modificados ao bel-prazer do usuário. Todavia, se você resolver modificá-lo, tome cuidado para não alterar a numeração das linhas, de maneira que o computador possa acessá-las convenientemente, sobretudo se houver novos módulos a serem adicionados.

O MENU PRINCIPAL

Ao ser executada, a versão completa do programa apresentará imediatamente algumas opções para entrada e saída de dados. Isso prepara o sistema para o uso de cassete ou acionador de discos. No caso do Spectrum, quem possuir um Microdrive pode adaptar o programa pa■■ utilizá-lo de acordo com as indicações oferecidas.

Se for de seu interesse, você pode dar entrada por um meio e saída por outro e mudar mais tarde.

O menu principal é mostrado e ■■ seguintes opções são apresentadas:

[C] ARREGAR
[G] RAVAR
[M] UDAR E/S
[E] DITAR
[L] IMPAR MEMÓRIA
[I] MPRIMIR
[S] AIR

CARREGAR

Se você quer continuar a trabalhar em um texto já existente, pressione **C** para iniciar ■ rotina que carrega o texto na memória. Qualquer outro aí presente será apagado. Portanto, uma mensagem "Confirme **(S/N)**" será inserida antes de se dar prosseguimento à rotina. Tecle S para continuar (ou N para interromper). Um nome de arquivo será então solicitado: ele é necessário para que se possa continuar. Em seguida, o referido arquivo é carregado na memória ■ pode ser trabalhado.

GRAVAR

A rotina de gravação é usada para criar arquivos seqüenciais para armazenamento dos textos. Quando essa opção for selecionada, teclando-se **G** — e desde que haja alguma coisa na memória — , será solicitado um nome para o arquivo. O Sinclair Spectrum gravará, mesmo que não haja nada ■■ memória. Uma entrada nula não será aceita. O sistema já "sabe" qual opção de **E/S** está sendo usada e continua a pedir informações de acordo com ■■ necessidades do cassete ou do acionador de discos. No final, pressione <**RETURN**> ou <**ENTER**> para efetuar a gravação.

Os usuários de disquete devem tomar bastante cuidado ao escolherem ■ nome do arquivo. Se o título de um arquivo já existente for usado, este será apagado, ficando no disco apenas o arquivo mais novo. Rotinas adicionais de gravação podem ser incorporadas a fim de proteger arquivos, trocar nomes etc.

ENTRADA/SAÍDA (E/S)

Se você quiser mudar ■■ parâmetros iniciais de entrada e saída (em inglês, abreviados por I/O de *input/output*), pode fazê-lo a qualquer momento, teclando **M**. Digite sua opção. Isso configura automaticamente o programa com as novas necessidades.

O programa do Sinclair Spectrum funciona com fita ou Microdrive. As mudanças necessárias para o Microdrive aparecem após ■ listagem principal.

MODO DE EDIÇÃO

Pressionar a tecla **E** leva você a um menu secundário que mostra:

[T] OPO
[F] IM
[P] RÓXIMA LINHA
[C] OR (não para o Apple e o TRS-Color)
[M] ENU

As duas primeiras opções o levam ▪ início (**TOPO**) ou ao fim (**FIM**) do texto que está na memória. A terceira opção conduz para a última linha com que se estava trabalhando. A última o faz retornar ▪ menu principal.

A opção **COR** permite ajustar as cores de frente, fundo e bordas, de acordo com a sua preferência.

Cada uma das três opções iniciais coloca o programa no modo de entrada. A tela mostra linhas demarcando o início e o fim do texto (pode aparecer apenas uma, se já houver algum texto armazenado na memória). Na parte mais baixa da tela há uma "área de trabalho" ▪ um indicador de "memória livre" (não disponível no micro Spectrum) que permite saber quantos caracteres disponíveis ainda se tem.

O texto é digitado na parte inferior da tela (área de trabalho). Os comandos **ENTER** ou **RETURN** o transferem para a memória e para a área de texto. O mesmo acontece quando duas linhas da tela são completadas.

Várias rotinas de edição estão embutidas no programa. Elas (e as teclas que as controlam) variam um pouco de máquina para máquina e serão discutidas em maior detalhe no momento oportuno. Toda a edição é feita na área de trabalho. Os controles de edição permitem que se movimente pela linha de texto na área de trabalho para inserção ou deleção (supressão) de caracteres, antes que o texto seja transferido para a memória. O texto que já estiver na memória — aquele que é mostrado no painel superior (área de texto) — deverá ser copiado, uma linha de cada vez, para a área de trabalho, antes de ser editado.

Novas linhas podem ser inseridas em qualquer ponto do texto. Isso se consegue pressionando a tecla apropriada para o modo "editor" (veja instruções detalhadas) e posicionando-se o marcador abaixo do ponto onde se quer inserir ▪ nova linha.

Da mesma forma, o texto pode ser removido, uma linha de cada vez, selecionando-se o modo editor e posicionando-se o marcador abaixo da linha a ser apagada antes de se pressionar ▪ tecla para deleção (supressão).

Teclas de controle permitem passear pelo texto, para ▪ frente ou para trás, dez linhas de cada vez (o Spectrum é uma exceção). Assim, o texto pode ser visualizado antes da impressão ou de realizar alguma alteração.

Pode-se voltar ao menu do modo de edição pressionando-se a tecla de "escape" designada. É possível também voltar do modo de edição para o menu principal sempre que seja necessário alterar algum parâmetro do sistema.

LIMPEZA DA MEMÓRIA

A memória é limpa por intermédio de outra opção disponível ▪ partir do menu principal. Mas esta só será oferecida depois que você responder sim à pergunta — "Confirma? (S/N)" — que se segue à teclagem de L. Note que essa op-

ção não redefine as opções de entrada e saída nem os parâmetros da impressora. Se você quiser reiniciar tudo, pressione S para retornar ao BASIC e, a seguir, **RUN**.

```
 10 POKE 23659,3
 20 BORDER 7: PAPER 7: INK 9:
  CLS
 30 DIM l(8): FOR N=1 TO 8:
 READ l(n): NEXT n
 40 LET Z$=""
100 LET ext=200: DIM t$(ext,32
): LET ll=32: LET pl=32
105 LET a$="
                "
110 LET t$(1)="ARQUIVO DO TOPO
DO TEXTO": LET t$(2)="--------
-----------------------": LET
t$(3)=a$
120 LET t$(4)=a$: LET t$(5)="-
--------------------------------
-": LET t$(6)="ARQUIVO DO FIM
 DO TEXTO"
130 LET t=1: LET b=6: LET p=4
140 CLS : PRINT  INVERSE 1;AT
0,8;" MENU PRINCIPAL "
150 PRINT AT 4,8;"1- Carregar
texto";AT 6,8;"2- Gravar texto
";AT 8,8;"3- Trocar papel";AT
10,8;"4- Editar";AT 12,8;"5- L
impar texto";AT 14,8;"6- Impri
mir texto";AT 16,8;"7- Alterar
impressora";AT 18,8;"8- Saida"
160 PRINT #1;TAB 7;"Selecione
opcao (1-8)"
170 LET a$=INKEY$: IF a$=""
THEN  GOTO 170
180 IF a$<"1" OR a$>"8" THEN
GOTO 170
190 LET a=VAL a$: CLS
200 GOSUB  l(a)
210 GOTO 140
500 CLS : PRINT  INVERSE 1;AT
4,8;" MENU - EDITOR "
510 PRINT AT 8,6;"1- Topo do t
exto";AT 10,6;"2- Fim do texto
";AT 12,6;"3- Proxima linha";
AT 14,6;"4- Sair do menu de ed
icao"
520 PRINT AT 18,7;"Selecione o
pcao (1-4)"
530 LET a$=INKEY$: IF a$=""
THEN  GOTO 530
540 IF a$<"1" OR a$>"4" THEN
GOTO 530
 550 LET a=VAL a$: CLS
 560 IF a=4 THEN  RETURN
 570 IF a=1 THEN  LET p=4
 580 IF a=2 THEN  LET p=b-2
 590 GOSUB 1000: GOSUB 2000
 600 GOTO 500
 900 PRINT AT 10,6;"Voce tem ce
 rteza?": PAUSE 0
 910 IF INKEY$="s" THEN  RUN
 920 RETURN
4000 RETURN
6000 REM carregar
6010 INPUT "Introduza o nome do
 arquivo", LINE n$: LOAD n$ DAT
A t$()
6020 LET b=VAL t$(1): LET t$(1)
="ARQUIVO DO TOPO DO TEXTO": RE
TURN
6200 REM gravar
6210 LET t$(1)=STR$ b
6220 INPUT "Introduza o nome do
 arquivo", LINE n$: IF n$="" OR
 LEN n$>10 THEN  GOTO 6220
6230 SAVE n$ DATA t$(): GOTO [illegible]
20
6500 INPUT AT 0,0;"Introduza a
largura do  formula-rio (1-80)"
,pl: IF pl<1 OR pl>80 THEN  GOT
O 6500
6510 INPUT AT 0,0;"Introduza o
numero de caracterespor linha
(1-";(pl);")";ll: IF ll<1 OR ll
>pl THEN  GOTO 6510
6520 LET ll=ll+1: RETURN
9000 DATA 6000,6200,3000,500,90
0,4000,6500,9999
```

T

```
10 CLS:POKE 150,1
20 PMODE 0:PCLEAR 1:CLEAR 15500
30 DIM TX$(500)
40 BL$=CHR$(128):TL=1:CP=1:MW=8
0:TW=60:PL=66:TH=60:GP=10:LF$=S
TRING$(3,13):GOSUB 5000
50 TX$(0)=STRING$(32,195)
60 TX$(TL)=STRING$(32,188)
70 CLS:PRINT @9,BL$;"menu";BL$;
"principal";BL$:PRINT @104,"(C)
ARREGAR":PRINT @136,"(G)RAVAR":
PRINT @168,"(M)UDAR ENTRADA/SAI
DA"
80 PRINT @200,"(E)DITAR TEXTO":
PRINT @232,"(L)IMPAR MEMORIA":P
RINT @264,"(I)MPRIMIR":PRINT @2
96,"(A)LTERAR IMPRESSORA":PRINT
 @328,"(S)AIR";
90 B$=INKEY$:IF B$="" THEN 90
100 B=INSTR("CGMELIAS",B$):IF B
=0 THEN 90
110 ON B GOSUB 4500,4000,5000,1
000,160,3000,5500,130
120 GOTO 50
130 CLS:PRINT"VOCE TEM CERTEZA
(S/N)?"
140 R$=INKEY$:IF R$<>"S" AND R$
<>"N" THEN 140
150 IF R$="S" THEN CLS:END ELSE
 RETURN
160 CLS:PRINT@8,BL$;"limpar";BL
$;"a";BL$;"memoria";BL$:PRINT:P
RINT"VOCE TEM CERTEZA? (S/N)"
170 B$=INKEY$:IF B$<>"N" AND B$
<>"S" THEN 170
180 IF B$="N" THEN RETURN
```

```
190 FOR K=1 TO TL:TX$(K)="":NEX
T:TL=1:CP=1:RETURN
1000 CLS:PRINT @42,BL$"menu"BL$
"de"BL$"edicao"BL$:PRINT @104,"
tOPO DO TEXTO":PRINT @168,"fIM
DO TEXTO":PRINT @232,"pROXIMA L
INHA":PRINT @296,"mENU PRINCIPA
L"
1010 B$=INKEY$:IF B$="" THEN 10
10
1020 B=INSTR("TFPM",B$):IF B=0
THEN 1010
1030 ON [illegible] GOTO 1050,1060,1070,1
080
1050 CP=1:GOTO 1070
1060 CP=TL
1070 GOSUB 2090:GOSUB 1500:GOTO
 1000
1080 RETURN
1500 A$=" "
1510 P=0:PRINT @384,A$
1520 CH=PEEK(1408+P):T$=INKEY$:
IF T$="" THEN CH=(CH+64) [illegible] 12
7:POKE 1408+P,CH:CH=(CH+64) AND
 127:POKE 1408+P,CH:GOTO 1520
1530 IF LEN(A$)=65 OR T$=CHR$(1
3) GOSUB 2000
1540 IF T$="^" THEN SF=0:GOSUB
2500:GOTO 1510
1550 IF P<LEN(A$)-1 [illegible] T$=CHR$
```

```
(10) THEN A$=LEFT$(A$,P)+MID$(A
$,P+2):GOTO 1600 ELSE IF T$=CHR
$(10) THEN 1520
1560 IF T$=CHR$(12) THEN RETURN
1570 IF T$=CHR$(21) THEN P=-(LE
N(A$)-1)*(P=0):GOTO 1600
1580 IF T$<>CHR$(8) AND T$<>CHR
$(9) AND ASC(T$)<32 THEN 1510
1590 IF T$<>"" AND T$<>CHR$(8)
[illegible] T$<>CHR$(9) THEN A$=LEFT$(A
$,P)+T$+MID$(A$,P+1):P=P+1
1600 PRINT @384,A$
1610 IF T$=CHR$(9) [illegible] P<LEN(A$
)-1 THEN P=P+1
1620 IF T$=CHR$(8) AND P>0 THEN
 P=P-1
1630 GOTO 1520
2000 X=1:IF LEN(A$)>33 THEN X=2
2010 FOR K=TL+X TO CP+X STEP -1
:TX$(K)=TX$(K-X):NEXT
2020 IF LEN(A$)>33 THEN TX$(CP)
=LEFT$(A$,32):TX$(CP+1)=MID$(A$
,33,LEN(A$)-33) ELSE TX$(CP)=A$
2030 FOR K=0 TO X-1
2040 IF RIGHT$(TX$(CP+K),1)=" "
 THEN TX$(CP+K)=LEFT$(TX$(CP+K)
,LEN(TX$(CP+K))-1):GOTO 2040
2050 NEXT
2060 A$=" ":P=0:PRINT @384,A$
2070 PRINT @416,""
2080 TL=TL+X:CP=CP+X
2090 IF CP<5 THEN ST=0 ELSE ST=
CP-5
2100 PRINT @0,;:FOR K=ST TO ST+
9:PRINT TX$(K);:IF LEN(TX$(K))<
32 THEN PRINT
2110 IF K=CP-1 THEN PRINT ">"
2120 NEXT:PRINT STRING$(32,140)
2130 PRINT @480,BL$;"mem";BL$;"
livre=";32*(501-TL);BL$;BL$;"cl
ear=menu";BL$;:POKE 1534,32:POK
E 1529,61:RETURN
```

```
10 CLS:KEY FF
20 COLOR 15,4,4:CLEAR 18000
30 DIM TX$(500):V$=STRING$(39,1
95)
40 TL=1:CP=1:MW=80:TW=60:PL=66:
TH=60:GP=10:L1$=CHR$(13):LF$=ST
RING$(3,L1$)
50 TX$(0)=STRING$(39,95):TX$(TL
)=TX$(0)
60 CLS:LOCATE 7,:PRINT"M E N U
 P R I N C I P A L":LOCATE 9,4:
PRINT"[C]arregar":LOCATE 9,6:PR
INT"[G]ravar"
70 LOCATE 9,8:PRINT"[E]ditar te
xto":LOCATE 9,10:PRINT"[L]impar
 memória":LOCATE 9,12:PRINT"[I]
mprimir":LOCATE 9,14:PRINT"[A]l
terar impressora":LOCATE 9,16:P
RINT"[S]air"
80 LOCATE 13,20:PRINT"Opção ?";
[illegible] B$=INKEY$:IF B$=""THEN 90
100 B=INSTR("CGELIAS",B$):IF B=
0 THEN 90
110 ON B GOSUB 4500,4000,1000,1
70,3000,5500,130
120 GOTO 50
130 CLS:BL$="Fim de programa":G
OSUB 220
140 LOCATE 12,10:PRINT"Confirme
 (S/N) ";
150 R$=INKEY$:IF R$="" THEN 150
160 IF R$="S" THEN CLS:END ELSE
 RETURN
170 CLS:BL$="Limpa a memória":G
OSUB 220
180 LOCATE 12,10:PRINT"Confirme
 (S/N) ";
190 R$=INKEY$:IF R$="" THEN 190
200 IF R$<>"S" THEN RETURN
210 FOR K=1 TO TL:TX$(K)="":NEX
T:TL=1:CP=1:RETURN
220 PRINTBL$:PRINT V$:RETURN
1000 CLS:BL$="Menu de edição":G
OSUB 220:LOCATE 9,8:PRINT"[T]op
o do texto":LOCATE 9,10:PRINT"[
F]im do texto":LOCATE 9,12:PRIN
T"[P]róxima linha":LOCATE 9,14:
PRINT"[M]enu principal"
1010 LOCATE 13,17:PRINT"Opção ?
";
1020 B$=INKEY$:IF B$="" THEN 10
20
1030 B=INSTR("TFPM",B$):IF B=0
THEN 1020
1040 [illegible] GOTO 1050,1060,1070,1
080
1050 CP=1:GOTO 1070
1060 CP=TL
1070 CLS:GOSUB 2080:GOSUB 1090:
GOTO 1000
1080 RETURN
1090 A$=" "
1100 P=0:LOCATE 0,16:PRINT A$;S
PC(80)
1110 LOCATE P+39*(P>38),16-(P>3
8):PRINTCHR$(219);
1120 T$=INKEY$:IF T$="" THEN 11
20
1130 IF LEN(A$)=79 OR T$=CHR$(1
3) THEN GOSUB 2000
1140 IF T$=CHR$(5) THEN SF=0:GO
SUB 2500:GOTO 1100
1150 IF P<LEN(A$)-1 [illegible] T$=CHR$
(4) THEN A$=LEFT$(A$,P)+MID$(A$
,P+2):GOTO 1200
1160 IF T$=CHR$(27) THEN [illegible]
1170 IF T$=CHR$(1) THEN P=0:GOT
O 1200 ELSE IF T$=CHR$(19) THEN
 P=LEN(A$)-1:GOTO 1200
1180 IF T$<>CHR$(28) [illegible] T$<>CH
R$(29) AND T$<CHR$(32) THEN 111
0
1190 IF T$<>"" AND T$<>CHR$(28)
 AND T$<>CHR$(29) THEN A$=LEFT$
(A$,P)+T$+MID$(A$,P+1):P=P+1
1200 LOCATE 0,16:PRINTA$;SPC(80
)
1210 IF T$=CHR$(28) AND P<LEN(A
$)-1 THEN P=P+1
1220 IF T$=CHR$(29) AND P>0 THE
N P=P-1
1230 GOTO 1110
2000 X=1:IF LEN(A$)>40 THEN X=2
2010 FOR K=TL+X TO CP+X STEP-1:
TX$(K)=TX$(K-X):NEXT
2020 IF LEN(A$)>40 THEN TX$(CP)
=LEFT$(A$,39):TX$(CP+1)=MID$(A$
,40,LEN(A$)-40) ELSE TX$(CP)=A$
2030 FOR K=0 TO X-1
2040 IF RIGHT$(TX$(CP+K),1)=" "
```

```
 THEN TXS(CP+K)=LEFTS(TXS(CP+K)
,LEN(TXS(CP+K))-1):GOTO 2040
2050 NEXT
2060 AS=" ":P=0:LOCATE 0,16:PRI
NTAS;SPC(80)
2070 TL=TL+X:CP=CP+X
2080 IF CP<9 THEN ST=0 ELSE ST=
CP-9
2090 LOCATE 0,0:FOR K= ST TO ST
+9:PRINTTXS(K):SPC(39-LEN(TXS(K
)))
2100 IF K=CP-1 THEN PRINTCHRS(1
75):SPC(38)
2110 NEXT:PRINTVS
2120 LOCATE 0,23:PRINT"Mem livr
e = ";39*(501-TL);"  ";TAB(26);
"<ESC>=Menu";:RETURN
```

```
10  HOME
20  DIM TXS(500)
30 TL = 1:CP = 1:MW = 80:TW = 6
0:PL = 66:TH = 60:GP = 10:LFS =
 CHRS (13) +  CHRS (13) +  CHR
S (13)
40 L1 = 6:S1 = L1:L2 = 1:S2 = L
2:D$ =  CHRS (4)
50  FOR Z = 1 TO 40:TXS(0) = TX
S(0) + "-":ASS = ASS + "*": NEX
T
60  GOSUB 5000
70 TXS(TL) = TXS(0)
80  HOME : HTAB 8: PRINT "M E [illegible]
 U   P R I N C I P A L"
90  VTAB 5: HTAB 10: PRINT "[C]
arregar": PRINT : HTAB 10: PRIN
T "[G]ravar": PRINT : HTAB 10:
PRINT "[M]udar E/S"
100  PRINT : HTAB 10: PRINT "[E
]ditar texto": PRINT : HTAB 10:
 PRINT "[L]impar memoria": PRIN
T : HTAB 10: PRINT "[I]mprimir"
: PRINT : HTAB 10: PRINT "[A]lt
erar impressora": PRINT : HTAB
10: PRINT "[S]air"
110  PRINT : PRINT : PRINT  TAB
( 13)"Opcao =>";
120  GET BS: IF BS =  CHRS (13)
 THEN 120
130 BBS = "CGMELIAS":B = 0: FOR
 Z = 1 TO 8: IF BS =  MIDS (BBS
,Z,1) THEN B = Z
140  NEXT : IF B = [illegible] THEN 120
150  [illegible] [illegible] GOSUB 4500,4000,5000,
1000,210,3000,5500,170
160  GOTO 70
170  HOME :BLS = "FIM DE PROGRA
MA": GOSUB 250
180  HTAB 12: VTAB 10: PRINT "C
ONFIRME (S/N)";: GET RS
190  IF RS <  > "S" THEN  RETUR
N
200  HOME : END
210  HOME :BLS = "LIMPAR A MEMO
RIA": GOSUB 250
220  HTAB 12: VTAB 10: PRINT "C
ONFIRME (S/N)";: GET RS
230  IF RS <  > "S" THEN  RETUR
N
240  FOR K = 1 TO TL:TXS(K) = "
": NEXT :TL = 1:CP = 1: RETURN
250  INVERSE : PRINT BLS;: PRIN
T  SPC( 40 -  LEN (BLS)): NORMA
L : RETURN
1000  HOME : PRINT  TAB( 12)"ME
[illegible] DE EDICAO"
1010  VTAB 9: HTAB 10: PRINT "[
T]OPO DO TEXTO"
1020  PRINT : PRINT  TAB( 10)"[
F]IM DO TEXTO"
1030  PRINT : PRINT  TAB( 10)"[
P]ROXIMA LINHA"
1040  PRINT : PRINT  TAB( 10)"[
M]ENU PRINCIPAL"
1050  PRINT : PRINT : PRINT  TA
B( 13)"Opcao =>";
1060  GET BS: IF BS =  CHRS (13
) THEN 1060
1070 BBS = "TFPM":B = 0: FOR Z
= 1 TO 4: IF BS =  MIDS (BBS,Z,
1) THEN [illegible] = [illegible]
1080  NEXT : IF [illegible] = [illegible] THEN 1060

1090  [illegible] B GOTO 1110,1120,1130,
1140
1110 CP = 1: GOTO 1130
1120 CP = TL
1130  HOME : GOSUB 2090: GOSUB
1500: GOTO 1000
1140  RETURN
1500 AS = "  "
1510 P = 1: HTAB 1: VTAB 20: CA
LL  - 958: PRINT  MIDS (AS,2);
1520  HTAB P: VTAB 20 + (P > 40
): GET TS
1530  IF  LEN (AS) = 82 OR TS =
  CHRS (13) THEN  GOSUB 2000
1540  IF TS =  CHRS (5) THEN  G
OSUB 2500: GOTO 1510
1550  IF TS =  CHRS (27) THEN
RETURN
1560  IF TS =  CHRS (1) THEN P
= 1: GOTO 1620
1570  IF TS =  CHRS (19) THEN P
 =  LEN (AS) - 1: GOTO 1620
1580  IF TS =  CHRS (4) AND (P
<  LEN (AS) - 1) THEN AS =  LEF
TS (AS,P) +  MIDS (AS,P + 2): [illegible]
OTO 1620
1590  IF TS =  CHRS (4) THEN 15
20
1600  IF TS <  >  CHRS (8) AND
TS <  >  CHRS (21) AND TS <  CH
RS (32) THEN 1520
1610  IF TS <  >  CHRS (8) AND
TS <  >  CHRS (21) THEN AS =  L
EFTS (AS,P) + TS +  MIDS (AS,P
+ 1):P = P + 1
1620  HTAB 1: VTAB 20: CALL  -
958: PRINT  MIDS (AS,2);
1630  IF TS =  CHRS (8) AND (P
> 1) THEN P = P - 1
1640  IF TS =  CHRS (21) AND (P
 <  LEN (AS) - 1) THEN [illegible] = [illegible] +
1
1650  GOTO 1520
2000 X = 1:AS =  MIDS (AS,2): I
F  LEN (AS) > 41 THEN X = [illegible]
2010  FOR K = TL + X TO CP + X
STEP  - 1:TXS(K) = TXS(K - X):
NEXT
2020  IF  LEN (AS) > 41 THEN TX
S(CP) =  LEFTS (AS,40):TXS(CP +
 1) =  MIDS (AS,41): GOTO 2040
```

Quais são as principais diferenças entre o editor de textos de *INPUT* e os editores vendidos [illegible] lojas especializadas?

A principal diferença consiste em que nosso editor é programado em BASIC, e os editores comerciais, em código de máquina. Tais editores atuam sobre uma "janela de texto", ou seja, permitem que se edite qualquer segmento de texto, acrescentando ou eliminando palavras, frases e parágrafos, por meio de simples movimentos do cursor na tela.

Em contraste com isso, nosso programa edita linha por linha do texto, exigindo que as linhas sejam escritas na parte inferior da tela e transferidas para o campo superior.

Esse modo de organizar o texto, junto com a lentidão inerente à linguagem BASIC, [illegible] a causa de outra diferença fundamental: [illegible] tempo de gravação [illegible] leitura dos textos em editores comerciais [illegible] muito menor.

Outro atributo dos editores comerciais é a hifenação automática das palavras. Somada ao alinhamento à direita, essa característica dá ao texto impresso um aspecto mais natural e profissional. Não há nenhum motivo, contudo, para que uma função de hifenação não possa ser incluída em nosso editor.

```
2030 TXS(CP) = AS: IF AS = " "
THEN 2070
2040  FOR K = 0 TO X - 1
2050  IF  RIGHTS (TXS(CP + K),1
) = " " THEN TXS(CP + K) =  LEF
TS (TXS(CP + K), LEN (TXS(CP +
K)) - 1): GOTO 2050
2060  NEXT
2070 AS = "  ":P = 1: HTAB 1: V
TAB 20: CALL  - 958: PRINT  MID
S (AS,2);
2080 TL = TL + X:CP = CP + X
2090  IF CP < 9 THEN ST = 0: GO
TO 2110
2100 ST = CP - [illegible]
2110  HTAB 1: VTAB 1: FOR [illegible] = S
T TO ST + 13: PRINT TXS(K);: IF
  LEN (TXS(K)) < 40 THEN  CALL
 - 868: PRINT
2120  IF [illegible] = CP - 1 THEN  CALL
 - 868: PRINT ">"
2130  NEXT : PRINT ASS;: PRINT
"memoria livre=";40 * (501 - TL
)
2140  RETURN
```

PROGRAMAÇÃO DE GRÁFICOS EM 3-D (1)

- O QUE É UM DESENHO TIPO **WIREFRAME**?
- COMO ORGANIZAR AS ROTINAS
- DESENHE UMA GRADE
- DESENHE UM CÍRCULO

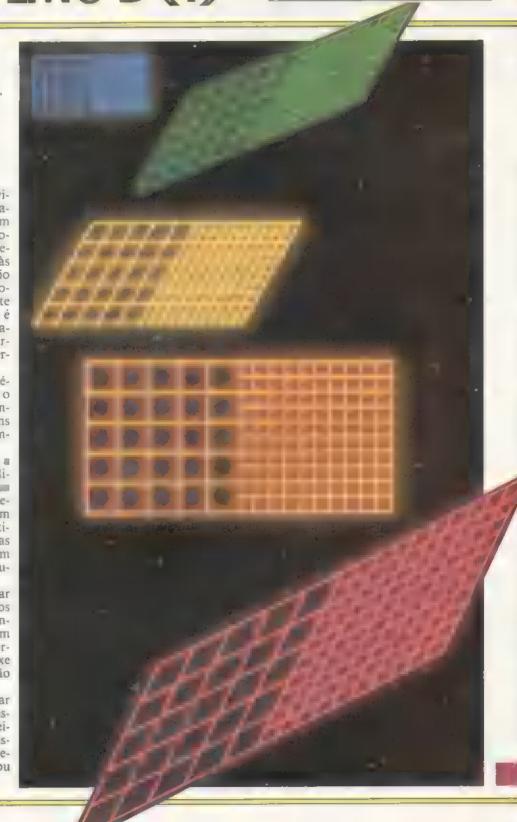
Neste primeiro artigo da série sobre desenho tipo wireframe, você aprenderá a elaborar grades e círculos. Mais tarde, verá como utilizá-los para estruturar imagens em 3-D.

Desenhos tridimensionais em movimento, do tipo *wireframe* (ou gradeado), têm se tornado imagens comuns em propagandas e, sem dúvida, impressionam muito. O efeito é uma das mais recentes contribuições do computador às comunicações visuais. Se, como usuário de um micro pessoal, você quiser produzir figuras como aquelas, certamente ficará desapontado ao verificar que é impossível obter o esplendor das imagens elaboradas por computadores, artistas e técnicos fotográficos nos comerciais de televisão.

Mas nem tudo está perdido: nesta série de artigos, você verá como extrair o máximo de seu computador, capacitando-o a produzir e movimentar imagens em 3-D de uma maneira que até engenheiros-desenhistas invejariam.

Como vimos nos artigos anteriores, a possibilidade de endereçar pontos individuais (pixels) na tela proporciona ao micro uma poderosa ferramenta de desenho. O TK-90X, por exemplo, tem 256 pixels na horizontal e 176 na vertical. Normalmente, desenhamos linhas retas e pontos que, combinados com recursos de cor, são usados para produzir imagens de alta resolução.

O método empregado para desenhar linhas em sistemas gráficos controlados por computador é, em geral, semelhante ao utilizado quando se trabalha com caneta e papel. Podemos mover o cursor pela tela fazendo com que ele deixe marcas ou não, e mudar de cor é tão simples quanto trocar de caneta.

A principal diferença entre desenhar com o computador ou manualmente está na rapidez da máquina e na perfeição com que ela traça linhas retas. Estas características tornam possível desenhar imagens em simples contornos ou mesmo wireframes tridimensionais.

O QUE É, AFINAL, UM WIREFRAME?

Wireframes são representações imaginárias formadas por grades de linhas sobre ■ superfície do objeto. Geralmente, neste tipo de desenho, ■ linhas de fundo do objeto também ficam aparentes, pois, para escondê-las, precisaríamos de muitos cálculos extras. O resultado é uma figura que parece feita somente de uma armação de arame.

Os wireframes podem ser movimentados ou rodados, ■ vemos em diversas propagandas. Nos micros pessoais, não ■ consegue ■ mesma rapidez. Para alcançar velocidade semelhante, precisaríamos que os quadros mudassem pelo menos ■ cada 1/25 de segundo. Na verdade, ■ maioria dos computadores usados nos comerciais de TV também não são tão rápidos assim, ■ as imagens parecem estar em "tempo real" graças a técnicas de filmagem.

Em desenhos estáticos, o tempo não é importante, embora ■ irritação esperar demais até que uma figura se complete. Mas, por outro lado, muitas vezes ■ é ■ interessante ver a figura sendo desenhada do que vê-la acabada, sobretudo ■ ela for bem complexa.

Começaremos desenhando formas simples, como cubo ou esfera; à medida que nos familiarizarmos com as técnicas do wireframe, tentaremos formas mais complicadas. Neste artigo, apren-

deremos a criar figuras bidimensionais que parecerão estar em espaço tridimensional. Infelizmente, as rotinas aqui apresentadas não servem para os micros das linhas ZX-81 e TRS-80.

PRIMEIROS PASSOS

A execução de um desenho complexo do tipo wireframe é bem demorada, independentemente da velocidade ou da capacidade de traçar linhas de um micro. Os primeiros passos consistem na incorporação de um conjunto de rotinas, que incluem os comandos gráficos básicos. Precisamos de um comando que mova o cursor sem desenhar e outro que o movimente desenhando. Estes comandos variam de máquina para máquina, como é o caso do **MOVE, DRAW** e **LINE**, já vistos em outros artigos.

Um programa gráfico geralmente começa ajustando o computador para o modo gráfico, se necessário, e limpando a tela. Devemos optar sempre pela resolução de tela mais alta. Convém ainda estruturar o programa de modo que todos os comandos gráficos sejam coletados juntos em sub-rotinas. Com essa medida, evitaremos reescrever desnecessariamente porções de programa que realizam a mesma tarefa. Além disso, tanto a expansão quanto o entendimento do programa serão mais simples se os comandos que realizam certa tarefa forem coletados numa mesma área. A estruturação de um programa deixa-o, de fato, um pouco lento; isso, porém, não tem maior importância quando lidamos com figuras estáticas.

INICIALIZAÇÃO DA MÁQUINA

Para começar o trabalho com as rotinas de desenho, digite esta parte do programa, mas não a rode ainda pois está incompleta.

```
9000  REM  INICIO
9005  PI = 4 *  ATN (1)
9010  HGR2 : HCOLOR= 3:N = 0
9070  RETURN
9100  REM  MOVE
9110  HPLOT X1,Y1
9120  RETURN
9200  REM  DESENHA
9210  HPLOT X1,Y1 TO X2,Y2
9220  RETURN
```

```
9000 PCLS
9030 RETURN
```

```
9000 REM INICIO
9010 BORDER 4: PAPER 7: INK 0:
CLS : LET N=0
9070 RETURN
9100 REM MOVE
9110 PLOT X,Y
9120 RETURN
9200 REM DESENHA
9210 DRAW X-PEEK 23677,Y-PEEK 2
3678
9220 RETURN
```

```
9000 CLS
9030 RETURN
```

Os programas do TRS-Color e MSX são mais curtos porque esses computadores permitem que movimentemos o cursor e desenhemos usando um simples comando. Os programas do Spectrum e do Apple, por sua vez, contêm sub-rotinas que movimentam o cursor sem desenhar (linhas 9100 a 9120) e que desenham na tela (linhas 9200 a 9220). Essas rotinas combinadas formam uma única rotina de movimento e desenho.

Agora, para fazer um desenho, não precisaremos dizer ao computador como mover o cursor ou marcar a tela: bastará definir a forma do objeto nos termos dessas rotinas básicas.

DESENHO DE LINHAS

A forma mais simples que um desenho pode assumir é, sem dúvida alguma, a de uma linha. Aqui está a rotina que faz isso; digite-a, mas não rode o programa ainda.

```
9500  REM  LINHA
9510 X1 = XS:Y1 = YS: GOSUB 910
0
9520 X2 = XE:Y2 = YE: GOSUB 920
0
9550  RETURN
```

```
9500 LINE(XS,YS)-(XE,YE),PSET
9550 RETURN
```

```
9500 REM LINHA
9510 LET X=XS: LET Y=YS: GOSUB
9100
9520 LET X=XE: LET Y=YE: GOSUB
9200
9550 RETURN
```

```
9500 LINE(XS,YS)-(XE,YE),15
9550 RETURN
```

A rotina especifica uma posição inicial — coordenadas (XS, YS) — e uma posição final — coordenadas (XE, YE) — para a linha. Em alguns computadores é necessário utilizar os comandos **DRAW** ou **LINE**; em outros, isto já está especificado nas rotinas das linhas 9100 a 9220. A rotina que desenha linhas é a base da maioria dos programas do tipo wireframe.

O DESENHO DA GRADE

Para representar uma superfície e qualquer irregularidade que possa ter — como morros, vales, fendas etc. — é melhor visualizá-la não como uma área contínua dentro de um retângulo, mas como uma grade de linhas horizontais ■ verticais. Assim, todas as irregularidades poderão ser representadas por distorções dessas linhas.

A próxima seção do programa define ■ rotina que desenha uma grade. Digite-a, mas não a rode.

```
5000 JA = LW / NX
5010 XS = XA
5020  FOR J = 0 TO NX
5025 YS = YA:XE = XS:YE = YA +
LH
5030  GOSUB 9500
5040 XS = XS + JA
5050  NEXT J
5060 JA = LH / NY
5070 YS = YA + LH
5080  FOR J = 0 TO NY
5090 XS = XA + LW:XE = XA:YE =
YS
5100  GOSUB 9500
5110 YS = YS - JA
5120  NEXT J
5130  RETURN
```

```
5000 JA=LW/NX
5010 XS=XA
5020 FOR JB=0 TO [illegible]
5025 YS=YA:XE=XS:YE=YA+LH
5030 GOSUB 9500
5040 XS=XS+JA
5050 NEXT JB
5060 JA=LH/NY
5070 YS=YA+LH
5080 FOR JB=0 TO NY
5090 XS=XA+LW:XE=XA:YE=YS
5100 GOSUB 9500
5110 YS=YS-JA
5120 NEXT JB
5130 RETURN
```

```
5000 LET JA=LW/NX
5010 LET XS=XA
5020 FOR J=0 TO NX
5025 LET YS=YA: LET XE=XS: LET
YE=YA+LH
5030 GOSUB 9500
5040 LET XS=XS+JA
5050 NEXT J
5060 LET JA=LH/NY
5070 LET YS=YA+LH
5080 FOR J=0 TO NY
5090 LET XS=XA+LW: LET XE=XA: L
ET YE=YS
5100 GOSUB 9500
5110 LET YS=YS-JA
5120 NEXT J
5130 RETURN
```

```
5000 JA=LW/NX
5010 XS=XA
5020 FOR JB=0 TO NX
5025 YS=YA:XE=XS:YE=YA+LH
5030 GOSUB 9500
5040 XS=XS+JA
5050 NEXT JB
5060 JA=LH/NY
5070 YS=YA+LH
5080 [illegible] JB=0 TO NY
5090 XS=XA+LW:XE=XA:YE=YS
5100 GOSUB 9500
5110 YS=YS-JA
5120 NEXT JB
5130 RETURN
```

As coordenadas (XA,YB) especificam o canto inferior esquerdo da grade. LW especifica a largura; LH, a altura; NX, o número de divisões horizontais, e NY, o número de divisões verticais. A variável JA determina a distância entre as linhas verticais, e o laço **FOR...NEXT** desenha horizontais, mantendo aquele intervalo. O segundo laço **FOR...NEXT** desenha verticais ■ intervalos calculados pela linha 5060.

A rotina entre as linhas 5000 e 5180 desenha as linhas horizontais da esquerda para a direita, e as verticais de baixo para cima, formando uma grade de superfície plana. Não poderíamos, portanto, representar irregularidades ■■ superfície dessa grade.

Para mostrá-la na tela, digite as linhas seguintes, que chamam a rotina, e rode o programa.

```
100  [illegible] 9000
175 XA = 0:YA = 0:LW = 279:LH =
 191:NX = 16:NY = 14
180  GOSUB 5000
190  STOP
```

```
100 PMODE 4:SCREEN 1,1
105 PI=4*ATN(1)
110 GOSUB 9000
175 XA=0:YA=0:LW=255:LH=191:NX=
4:NY=3
180 GOSUB 5000
190 GOTO 190
```

```
100 GOSUB 9000
175 LET XA=0: LET YA=0: LET LW
=255: LET LH=175: LET NX=16:
LET NY=12
180 GOSUB 5000
190 STOP
```

```
100 SCREEN 2:COLOR 1,9,10
105 PI=4*ATN(1)
110 GOSUB 9000
175 XA=0:YA=0:LW=255:LH=191:NX=
4:NY=3
180 GOSUB 5000
190 GOTO 190
```

Ao rodar o programa, aparecerá uma grade ocupando toda a tela. Faça as mudanças abaixo para observar a versatilidade do programa:

```
175 XA = 10:YA = 10:LW = 240:LH
 = 144:NX = 1:NY = 1
```

```
175 XA=10:YA=10:LW=240:LH=160:N
X=1:NY=1
```

```
175 LET XA=10: LET YA=10: LET
LW=240: LET LH=144: LET NX=1:
LET NY=1
```

```
175 XA=10:YA=10:LW=240:LH=160:N
X=1:NY=1OK
```

Desta vez, aparece na tela uma caixa retangular, já que se especificou uma grade com apenas uma divisão horizontal ■ uma vertical. Fornecendo valores adequados ■ NX e NY, como acima, podemos construir uma grade com números de linhas horizontais diferente do de verticais. Faça as mudanças que se seguem e rode o programa:

```
175 XA = 0:YA = 0:LW = 180:LH =
 130:NX = 16:NY = 16
```

```
175 XA=0:YA=31:LW=160:LH=160:NX
=15:NY=10
```

```
175 LET XA=0: LET YA=0: LET LW
=160: LET LH=144: LET NX=15:
LET NY=10
```

```
175 XA=0:YA=31:LW=160:LH=160:NX
=15:NY=10
```

Para a forma quadrada, determinamos os valores apropriados de LW e LH e ■ número de divisões por NX e NY.

O DESENHO DE CÍRCULOS

Os círculos também são muito úteis nos desenhos tipo wireframe. Em alguns

micros, eles podem ser executados diretamente por um comando, bastando que se definam o centro e o raio.

O comando direto, contudo, não nos oferece o grau de controle necessário para ■ elaboração de um desenho tridimensional. Em perspectiva, um círculo visto de um certo ângulo pode parecer uma elipse ou até uma curva. Embora desenhe elipses, o comando **CIRCLE** não é capaz de lhes conferir a tridimensionalidade essencial para garantir a aparência realística ao objeto. Convém, portanto, definir uma função genérica.

Podemos compor um círculo usando uma série de segmentos de reta. Se estes forem suficientemente pequenos, a curva parecerá contínua. Mas, quanto menores forem os segmentos de reta, maior será o número deles e mais prolongado ■ tempo de execução. Aqui está uma rotina que desenha um círculo de raio R com centro em (XS, YS). Digite-a, mas não ■ rode ainda.

```
6000  IF N = 0 THEN N = 20 +  I
NT (R / 10)
6020  JA = ■ * PI / N
6050  XR = XS:YR = YS
6060  JB = 0:XS = XS + R
6070   FOR J = 2 TO N
6080  JB = JB + JA
6090  XE = XR + R *  COS (JB):YE
 = YR + R *  SIN (JB): GOSUB 95
00
6100  XS = XE:YS = YE
6110   NEXT J
6120  XE = XR + R:YE = YR: GOSUB
 9500
6130  XS = XR:YS = YR
6160   RETURN
```

```
6000 IF N=0 THEN N=20+INT(R/10)
6020 JA=2*PI/N
6050 XR=XS:YR=YS
6060 JB=0:XS=XS+R
6070 FOR JC=2 TO N
6080 JB=JB+JA
6090 XE=XR+R*COS(JB):YE=YR+R*SI
N(JB):GOSUB 9500
6100 XS=XE:YS=YE
6110 NEXT JC
6120 XE=XR+R:YE=YR:GOSUB 9500
6130 XS=XR:YS=YR
6160 RETURN
```

```
6000 IF N=0 THEN  LET N=20+INT
(R/10)
6020 LET JA=2*PI/N
6050 LET XR=XS: LET YR=YS
6060 LET JB=0: LET XS=XS+R
6070 FOR J=2 TO N
6080 LET JB=JB+JA
6090 LET XE=XR+R*COS JB: LET YE
=YR+R*SIN JB: GOSUB 9500
6100 LET XS=XE: LET YS=YE
6110 NEXT J
6120 LET XE=XR+R: LET YE=YR: GO
SUB 9500
6130 LET XS=XR: LET YS=YR
6160 RETURN
```

```
6000 IF N=0 THEN N=20+INT(R/10)
6020 JA=2*PI/N
6050 XR=XS:YR=YS
6060 JB=0:XS=XS+R
6070 FOR JC=2 TO N
6080 JB=JB+JA
6090 XE=XR+R*COS(JB):YE=YR+R*SI
N(JB):GOSUB 9500
6100 XS=XE:YS=YE
6110 NEXT JC
6120 XE=XR+R:YE=YR:GOSUB 9500
6130 XS=XR:YS=YR
6160 RETURN
```

A variável N especifica o número de segmentos de reta que serão usados na composição do círculo. Caso façamos N = 0, a linha 6000 calculará o número de segmentos necessários para fazer o círculo mais liso, levando em conta, é claro, o tamanho dele.

A linha 6020 calcula o ângulo de cada segmento de reta na circunferência. A linha 6050 move o cursor para uma posição na circunferência. O laço **FOR...NEXT** traça todos ■ segmentos, com exceção do último, que é desenhado pela linha 6120, para garantir que se una ao primeiro.

Para ver ■ rotina funcionando, apague a linha 180 e acrescente estas:

```
150  FOR R = 20 TO 70 STEP 10
155  XS = 128:YS = 102:N = 24
160   GOSUB 6000
170   NEXT ■
```

```
150 FOR R=0 TO 100 STEP 20
155 XS=128:YS=102:N=24
160 GOSUB 6000
170 NEXT R
```

```
150 FOR R=20 TO 70 STEP 10
155 LET XS=128: LET YS=102:
LET N=24
160 GOSUB 6000
170 NEXT ■
```

```
150 FOR R=0 TO 100 STEP 20
155 XS=128:YS=102:N=24
160 GOSUB 6000
170 NEXT R
```

Posso colorir wireframes?

Desenhos do tipo wireframe geralmente são apresentados em duas cores, sobretudo preto e branco.

A presença de muitas cores tende a complicar a imagem, muitas vezes anulando o efeito tridimensional. Além disso, ■ adição de cores ■ um desenho complexo esbarra em limitações da própria máquina.

No TRS-Color, o maior número de cores sacrifica a alta resolução gráfica, pois a resolução se reduz à metade quando passamos de um modo de duas cores para um de quatro.

No Spectrum e no MSX, determinadas porções da tela só podem apresentar duas cores. No Spectrum, esta porção corresponde ■ cada bloco de oito por oito pontos da tela; no MSX, a cada conjunto de oito pontos adjacentes na horizontal.

Já no Apple e no TK-2000, algumas cores ocupam colunas pares, outras ímpares, em alta resolução. Se colocarmos pontos ou linhas em colunas inadequadas à sua cor, eles simplesmente não aparecerão na tela.

Rodado o programa, um certo número de círculos aparecerá no centro da tela. Como na rotina da grade, podemos variar os parâmetros no programa que chama a rotina ■ obter uma disposição de círculos diferente. A linha 150 ajusta o raio do primeiro círculo e determina o quanto ele aumenta. A 155 define o centro do círculo e o número de segmentos de reta que formarão a circunferência. Como exercício, varie esses valores e veja os resultados.

Talvez você queira saber o que aconteceu com ■ rotina da grade: ela ainda está na memória, mas, como reescrevemos as linhas de código que chamam a rotina, o computador não as mostra. Rotinas como esta podem ser guardadas num arquivo de rotinas gráficas para uso posterior. Elas terão utilidade em vários tipos de programa gráfico, bem como nos de wireframes. Na medida em que for preciso, adicione novas rotinas. Uma vez no computador, salve-as em cassete ou disco e carregue-as novamente na memória quando for utilizá-las. Junte-as, se necessário. No próximo artigo desta série, veremos como usar essas rotinas na criação de wireframes tridimensionais.

UM EDITOR DE TEXTOS (2)

Adicionar, corrigir, apagar, tudo ■ possível com um programa editor. Portanto, ■ você quer obter um texto perfeito, comece com um rascunho e melhore-o ■ vontade.

Neste artigo, você encontrará a parte principal do programa editor de textos e ■ explicações sobre o uso das funções de edição em cada máquina. Embora ■ procedimentos sejam muito parecidos, controles específicos e algumas características variam.

Após digitar as linhas aqui apresentadas, você já poderá usar todas ■ funções de edição do programa. Mas lembre-se de que só será possível imprimir o texto criado depois da última parte, que daremos ■ próximo artigo.

Como você verá, ■ opções de edição oferecidas pelo programa são úteis em vários níveis. O mais simples consiste na correção dos erros de grafia. É facílimo eliminá-los: basta colocar sua "caneta eletrônica", o cursor, sobre ■ palavra errada ■ inserir ou apagar ■ letras que quiser.

Se ■ ortografia não é um problema para você, ■ sim a composição do texto, este programa ■ revelará ideal. Ao escrever algo importante, como uma solicitação de emprego ou uma explicação ao gerente sobre um problema em sua conta bancária, você já não precisará gastar uma pilha de folhas de papel, riscando ■ reescrevendo até encontrar ■ termo mais preciso, ou a forma mais adequada.

Vá direto ■ computador e economize papel e paciência.

Comece digitando um rascunho do texto e, depois, melhore-o. Analise o que já escreveu "passeando" pelo texto, do início ■ fim, quantas vezes julgar necessário. Quando decidir fazer alguma modificação, corrija, apague ou introduza o que quiser. Trabalhe ■ vontade, até que ■ texto lhe agrade.

Uma outra possibilidade que o programa lhe oferece é a de armazenar o texto elaborado, para usá-lo em outra oportunidade ou, apenas, para saber mais tarde ■ que escreveu.

Os recursos disponíveis neste programa são semelhantes, embora muito mais simples, ■ dos mais modernos processadores de texto usados, por exemplo, para produzir ■ que você está lendo agora. Quem já viu como tudo isso era feito há pouco tempo (e ainda é, em alguns lugares) não terá dúvidas a respeito das vantagens de utilizar um programa editor.

S

O programa do Spectrum foi elaborado para uso com um gravador cassete. Se você tem um Microdrive, faça as modificações necessárias.

As funções de edição são quase idênticas às do BASIC e, portanto, não é necessário um editor especial. Todas as entradas de texto, bem como as alterações, devem ser feitas na parte inferior da tela, ■ área de trabalho, ficando a parte superior para ■ visualização do que já foi escrito.

Durante a edição de linhas na área de trabalho, pressione **<CAPS SHIFT>** ■ 5 para mover o cursor para ■ esquerda e **<CAPS SHIFT>** e ■ para movê-lo para ■ direita. Quando ele estiver na posição adequada, digite os caracteres necessários. Para apagar, coloque o cursor à direita do caractere e pressione **<CAPS SHIFT>** ■ 0.

Para transferir ■ texto para a memória e para a área superior, tecle **<ENTER>**. Linhas com mais de 64 caracteres irão automaticamente para a memória, visto que a área de trabalho su-

- CORREÇÃO DE ERROS ORTOGRÁFICOS
- COMPOSIÇÃO
- DO RASCUNHO AO TEXTO FINAL
- ANÁLISE SEM PRESSA
- ARMAZENAGEM DO TEXTO
- AJUSTE DO TAMANHO E ACERTO DOS PARÁGRAFOS
- AS FUNÇÕES DE EDIÇÃO EM CADA MÁQUINA

porta no máximo duas linhas de texto. Mas elas permanecerão juntas para efeito de impressão, a não ser que se usem comandos de formatação.

As teclas de cursor para cima e para baixo são usadas para localizar linhas do texto na memória. O marcador fica abaixo da linha de interesse, na área de visualização do texto. Esta linha pode ser copiada para a área de trabalho usando-se a função **EDIT** habitual — pressione **<SHIFT>** e l. Para apagar a linha, pressione **<CAPS SHIFT>** e 9, deixando o modo editor e, em seguida, **<CAPS SHIFT>** e **<SYMBOL SHIFT>** simultaneamente.

Para ler todo o texto da memória, use as teclas de cursor para cima e para baixo ou **<CAPS SHIFT>** e 6 e **<CAPS SHIFT>** e 7. Estas teclas moverão o texto para cima e para baixo, uma linha de cada vez.

```
1000 REM imprime a tela
1005 PLOT 0.13: DRAW 255,0: PLO
T 0,14: DRAW 255.0
1010 PRINT AT 0,0;: FOR n=p-10
TO p+8
1020 IF n<1 OR n>200 THEN  PRIN
T s$: GOTO 1050
1025 IF n=p THEN  PRINT
1030 PRINT t$(n)
1040 POKE 22528+320.120
1050 NEXT n
1060 RETURN
2000 REM entrada
2010 LET i$="": LET j$=""
2015 PRINT #1;AT 0,0;i$: FLASH
1; BRIGHT 1;" ": FLASH 0: BRIGH
T 0;j$;" "
2020 PAUSE 0: LET a$=INKEY$: IF
 a$="" THEN  GOTO 2020
2025 SOUND .01.20
2030 IF a$<CHR$ 32 THEN  GOSUB
2500
2040 IF a$>CHR$ 31 AND a$<CHR$
123 THEN  LET i$=i$+a$
2042 IF a$=CHR$ 13 AND b=ext-6
THEN  PRINT #1;AT 0,0;s$;s$: FL
ASH 1;"ARQUIVO LOTADO": SOUND 2
,10: RETURN
2045 IF a$=CHR$ 13 OR LEN i$+LE
N j$=64 THEN  PRINT #1;AT 0,0;s
$;s$;s$: LET i$=i$+j$: GOTO 210
0
2050 IF a$=CHR$ 14 THEN  RETURN

2052 IF a$=CHR$ [illegible] THEN  INPUT "
Introduza palavra procurada", L
INE z$: IF z$="" THEN  GOTO 205
2
2053 IF a$=CHR$ [illegible] THEN  LET p=4
: GOSUB 8000
2054 IF a$=CHR$ [illegible] THEN  GOSUB 8
000
2055 IF a$=CHR$ 5 THEN  GOSUB [illegible]
500
2060 GOTO 2015
2100 IF LEN i$>32 THEN  GOTO 21
50
2105 FOR n=b+1 TO p STEP -1
2110 LET t$(n+1)=t$(n)
2120 NEXT n
2130 LET t$(n+1)=i$: LET p=p+1:
 LET b=b+1
2140 GOSUB 1000: GOSUB 2500: GO
TO 2000
2150 FOR n=b+1 TO p STEP -1
2160 LET t$(n+2)=t$(n): LET t$(
n+3)=t$(n+1)
2170 NEXT n
2180 LET t$(n+1)=i$( TO 32): LE
T t$(n+2)=i$(33 TO ): LET p=p+2
: LET b=b+2
2190 GOTO 2140
2200 LET p=p-1: FOR n=p TO b+1
2210 LET t$(n)=t$(n+1)
2220 NEXT n
2225 LET b=b-1
2230 GOSUB 1000
2240 RETURN
2500 REM codigos de controle
2520 IF a$=CHR$ 10 AND p<b-2 TH
EN  LET p=p+1: GOSUB 1000
2530 IF a$=CHR$ 11 AND p>t+3 TH
EN  LET p=p-1: GOSUB 1000
2540 IF a$=CHR$ 12 AND LEN i$>0
 THEN  LET i$=i$( TO LEN i$-1)
2550 IF a$=CHR$ 8 AND LEN i$>0
THEN  LET j$=i$(LEN i$)+j$: LET
 i$=i$( TO LEN i$-1)
2560 IF a$=CHR$ 9 AND LEN j$>0
THEN  LET i$=i$+j$(1): LET j$=j
$(2 TO )
2570 IF a$<>CHR$ 7 THEN  GOTO 2
580
2572 LET j$=t$(p-1): LET i$="":
 PRINT #1;AT 0,0;s$;s$
2575 IF j$(LEN j$)=CHR$ 32 THEN
  LET j$=j$( TO LEN j$-1): IF L
EN j$>0 THEN  GOTO 2575
2580 IF a$=CHR$ 15 AND p>4 THEN
  GOSUB 2200
2690 RETURN
3000 REM cores
3010 PRINT AT 10,4;"Selecione c
or de fundo (0-7)"
3020 PAUSE 0: LET a$=INKEY$: IF
 a$<"0" OR a$>"7" THEN  GOTO 30
20
3030 PAPER VAL a$: BORDER VAL a
$: CLS : RETURN
```

T

Em geral, o texto é exibido na tela (e eventualmente impresso) em letras maiúsculas. Minúsculas podem ser usadas teclando **<SHIFT>** 0. Na tela, aparecerão como caracteres invertidos (fundo escuro, caractere claro).

Os controles de edição seguem as diretrizes anteriormente mencionadas. Edita-se o texto na área de trabalho, localizada na parte inferior da tela. A parte superior é utilizada para mostrar o

texto que já está na memória. As teclas de controle do cursor têm um papel importante na edição. As teclas <→> e <←> permitem que você mova o cursor pelo texto na área de trabalho. Se você pressionar **<SHIFT>** juntamente com <←>, o cursor pulará para o início ou fim da linha.

Para inserir caracteres em um ponto da linha na área de trabalho, coloque o cursor à direita do local escolhido e tecle o que desejar. Não se pode sobrepor caracteres. Para eliminar erros, coloque o cursor sobre ■ caractere e tecle a seta para baixo.

Para ativar o modo editor, tecle a seta para cima. O cursor vai para a última posição acessada, indicada por um sinal > piscando.

No modo editor, pode-se inspecionar o texto da memória, rolando-o para cima e para baixo com as setas respectivas. Avanços (ou retrocessos maiores) são possíveis com as teclas U e D, que pulam de 10 em 10 linhas.

Se quiser apagar linhas de texto, posicione o marcador logo abaixo da linha e pressione **<SHIFT>** e seta para baixo, ao mesmo tempo. Linhas em branco podem ser inseridas a partir da área de trabalho, pressionando-se **<ENTER>** quando ela está vazia.

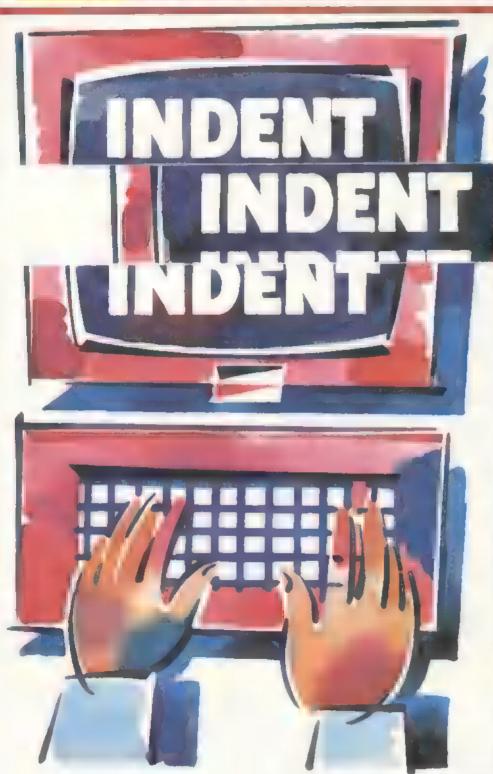
Para copiar uma linha da memória para a área de trabalho, coloque o marcador logo abaixo dela ■ pressione **C** (dentro do modo editor). A linha modificada não substitui a original, devendo esta ser apagada depois. Volta-se do modo editor para a área de trabalho teclando-se **<ENTER>**. **<CLEAR>** retorna da área de trabalho para o menu de edição.

```
2500 PM=5:IF CP<5 THEN PM=CP
2510 TB$=INKEY$:IF TB$="" THEN
PRINT @PM*32,"":PRINT @PM*32,">
":GOTO 2510
2520 CP=CP+(TB$="^")-(TB$=CHR$(
10))+10*((TB$="U")-(TB$="D"))
2530 IF CP<1 THEN CP=1
2540 IF CP>TL THEN CP=TL
2550 GOSUB 2090
2560 IF TB$=CHR$(13) THEN RETUR
N
2570 IF CP>1 AND TB$=CHR$(91) T
HEN TL=TL-1:FOR K=(CP-1) TO TL:
TX$(K)=TX$(K+1):NEXT:TX$(TL+1)=
"":CP=CP-1:GOSUB 2090
2580 IF CP>1 AND TB$="C" THEN F
OR K=32 TO 1 STEP -1:IF MID$(TX
$(CP-1),K,1)="" THEN NEXT ELSE
A$=LEFT$(TX$(CP-1),K)+"":RETURN
2590 IF TB$="P" GOSUB 5070
2600 IF TB$="@" THEN SF=SF+1:IF
 SF=1 THEN SS=CP ELSE SE=CP:SF=
0:GOSUB 5130
2610 GOTO 2500
3000 RETURN 'LINHA TEMPORARIA
4000 CLS:IF TL=1 THEN PRINT @7,
"nada para guardar"FOR Z=1 TO 1
000:NEXT:RETURN
4010 CLS:LINEINPUT" NOME DO ARQ
UIVO?";F$
4020 IF LEFT$(F$,1)<"A" OR LEFT
$(F$,1)>"Z" THEN 4010
4030 IF TS=1 THEN 4120
4040 CLS:MOTORON:AUDIO ON:PRINT
"POSICIONE O TAPE E PRESSIONE
 <ENTER>"
4050 IF INKEY$<>CHR$(13) THEN 4
050 ELSE MOTOROFF:AUDIO OFF:PRI
NT"POSICIONE O GRAVADOR EM 'REC
' E PRESSIONE <ENTER>"
4060 IF INKEY$<>CHR$(13) THEN 4
060
4070 MOTORON:FOR K=1 TO 1000:NE
XT:OPEN"O",#-1,F$
4080 PRINT #-1,CP,TL
4090 FOR K=1 TO TL-1:PRINT #-1,
TX$(K):NEXT
4100 CLOSE #-1
4110 RETURN
4120 CLS:PRINT"CERTIFIQUE-SE DE
 QUE O DRIVE    ESTA LIGADO E O
 DISCO INSERIDO. PRESSIONE <ENT
ER>PARA CONTINUAR"
4130 IF INKEY$<>CHR$(13) THEN ■
130 ELSE F$=F$+"/DAT"
4140 OPEN  "O", #1, F$
4150 WRITE #1,CP,TL
4160 FOR K=1 TO TL-1
4170 WRITE #1,TX$(K)
4180 NEXT:CLOSE #1:RETURN
4500 CLS:PRINT @8,BL$;"carregar
";BL$;"um";BL$;"arquivo";BL$:IF
 TL=1 THEN 4540
4510 PRINT "VOCE TEM CERTEZA (S
/N)?"
4520 R$=INKEY$:IF R$<>"S" AND R
$<>"N" THEN 4520
4530 IF R$="N" THEN RETURN
4540 CLS:LINEINPUT"NOME DO ARQU
IVO: ":F$
4550 IF LEFT$(F$,1)< "A" OR LEF
T$(F$,1)>"Z" THEN 4540
4560 IF DL=1 THEN 4645
4570 MOTORON:AUDIOON:PRINT"POSI
CIONE O TAPE EM 'PLAY' E    PRE
SSIONE <ENTER>"
4580 IF INKEY$<>CHR$(13) THEN 4
580
4590 OPEN "I",#-1,F$
4600 INPUT #-1,CP,TL
4610 FOR K=1 TO TL-1:INPUT #-1,
TX$(K):NEXT
4620 CLOSE #-1:GOSUB 2090
4630 TX$(TL)=STRING$(32,126)
4640 RETURN
4645 F$=F$+"/DAT"
4650 OPEN "I", #1, F$:INPUT #1,
CP,TL
4660 FOR K=1 TO TL-1:INPUT #1,T
X$(K)
4670 NEXT:CLOSE #1:RETURN
5000 CLS:PRINT @9,BL$;"selecao"
;BL$;"e";CHR$(124);"s";BL$:PRIN
T @96,"CARREGAR DE (F)ITA OU (D
)ISCO?";
5010 B$=INKEY$:IF B$<>"F"  AND
B$<>"D" THEN 5010
5020 PRINT B$:DL=0:IF B$="D" TH
EN DL=1
5030 PRINT:PRINT"GRAVAR ■■ (F)I
TA OU (D)ISCO?";
5040 B$=INKEY$:IF B$<>"F" AND B
$<>"D" THEN 5040
5050 PRINT B$:TS=0:IF B$="D" TH
■■ TS=1
5060 RETURN
5070 RETURN 'LINHA TEMPORARIA
5130 RETURN 'LINHA TEMPORARIA
5500 CLS:PRINT @3,BL$;"preparac
ao";BL$;"da";BL$;"impressora";B
L$
5510 PRINT @128,;:INPUT"LARGURA
 DO FORMULARIO";MW:MW=INT(MW):I
F MW<1 THEN 5510
5520 INPUT "COMPRIMENTO DE LINH
A";TW:TW=INT(TW):IF TW<1 OR TW>
MW THEN 5520
5530 INPUT"COMPRIMENTO DO FORMU
LARIO";PL:PL=INT(PL):IF PL<1 TH
EN 5530
5540 INPUT"LINHAS POR PAGINA";T
H:TH=INT(TH):IF TH<1 OR TH>PL T
HEN 5530
5550 GP=INT((MW-TW)/2):LF$=STRI
NG$(INT((PL-TH)/2),13)
5560 PRINT:PRINT:PRINT" CONFIRM
A (S/N)?"
5570 R$=INKEY$:IF R$<>"N" AND R
$<>"S" THEN 5570
5580 IF R$="S" THEN RETURN ELSE
 5500
```

Os editores para o MSX e o Apple seguem, em linhas gerais, o que já foi dito. É importante prestar atenção quando se usam minúsculas, pois as respostas requeridas pelo computador só são aceitas em maiúsculas. Por outro lado, dentro do editor, podem-se usar sem problemas ■■ sinais de pontuação ■ acentuação (esta, no caso do MSX).

Trabalha-se o texto na área situada na parte inferior da tela. Quando se entra nesta área, o cursor é visível no começo na linha. A partir daí, escreve-se normalmente. As setas para a direita ■ para ■ esquerda movimentam o cursor nas respectivas direções. Pode-se acelerar sua movimentação por meio de **<CTRL>** **<A>** ■ **<CTRL>** **<S>**. A pressão simultânea dessas teclas coloca o cursor na primeira e última posição da linha, respectivamente. Para inserir um caractere, leve o cursor para a posição imediatamente à direita de onde fará a inserção e digite o caractere. Não é possível sobrepor caracteres. Para apagar algo errado, coloque o cursor sobre o caractere e digite **<CTRL>** **<D>**.

Para ativar o modo editor, pressione **<CTRL>** **<E>**. O marcador do texto começará a piscar. Teclando **<CTRL>** **<O>** e **<CTRL>** **<Z>**, você poderá inspecionar o texto todo. Esses comandos fazem com que o marcador (e ■ texto) ro-

lem para cima ou para baixo. **<CTRL> <W>** e **<CTRL> <X>** provocam saltos maiores (de 10 linhas).

Para apagar linhas de texto, estando no modo editor, pressiona-se **<CTRL> <L>**. A linha logo acima do marcador será apagada.

Quando quiser transferir uma linha de memória para a área de trabalho, tecle **<CTRL> <C>**, com o marcador embaixo da linha desejada. A linha aparecerá na área de trabalho e poderá ser editada imediatamente. Note que, ao teclar **<RETURN>** para devolvê-la para a memória, ela não substituirá a original. É necessário apagá-la.

Linhas em branco podem ser criadas simplesmente teclando-se **<RETURN>**, com a área de trabalho vazia.

O programa para o Apple armazena e grava dados somente em disco. Já o programa para o MSX trabalha apenas com gravador cassete. Não estranhe se a leitura de dados no Apple for morosa. Para não haver problema com os sinais de pontuação, é necessário que os caracteres sejam lidos um de cada vez. Se você quiser mais velocidade, não se importando com a pontuação, elimine o laço das linhas 4600 e 4610 e deixe apenas a instrução **INPUT TX$ (K)**.

```
2500 PM=9:IF CP<9 THEN PM=CP
2510 TB$=INKEY$:IF TB$=""THEN L
OCATE 0,PM:PRINTCHR$(219);:PRIN
TCHR$(8);CHR$(175);:GOTO 2510
2520 CP=CP+(TB$=CHR$(17))-(TB$=
CHR$(26))+10*((TB$=CHR$(23))-(T
B$=CHR$(24)))
2530 IF CP<1 THEN CP=1
2540 IF CP>TL THEN CP=TL
2550 GOSUB 2080
2560 IF TB$=CHR$(27) THEN RETUR
N
2570 IF CP>1 AND TB$=CHR$(12) T
HEN TL=TL-1:FOR K=CP-1 TO TL:TX
$(K)=TX$(K+1):NEXT:TX$(TL+1)=""
:CP=CP-1:GOSUB 2080
2580 IF CP>1 AND TB$=CHR$(3) TH
EN A$=TX$(CP-1)+" ":RETURN
2590 IF TB$=CHR$(16) THEN GOSUB
 5000
2600 IF TB$=CHR$(15) THEN SF=SF
+1:IF SF=1 THEN SS=CP ELSE SE=C
P:SF=0:GOSUB 5060
2610 GOTO 2500
3000 RETURN 'LINHA TEMPORARIA
4000 CLS:BL$="Gravar na fita":G
OSUB 220:IF TL=1 THEN BEEP:LOCA
TE 12,11:PRINT"Nada a gravar!":
FOR Z=1 TO 1000:NEXT:RETURN
4010 LOCATE 4,4:LINEINPUT"Nome
do arquivo? ";F$
4020 IF LEFT$(F$,1)<"A" OR LEFT
$(F$,1)>"Z"THEN 4010
4030 LOCATE 1,6:PRINT"Posicione
 a fita e pressione <RETURN>"
4040 IF INKEY$<>CHR$(13) THEN 4
040 ELSE PRINT:PRINT"Deixe o gr
avador pronto para iniciar    g
ravação e pressione <RETURN>"
4050 IF INKEY$<>CHR$(13) THEN 4
050
4060 OPEN F$ FOR OUTPUT AS #1
4070 PRINT#1,CP,TL
4080 FOR K=1 TO TL-1:PRINT#1,TX
$(K):NEXT
4090 CLOSE #1
4100 RETURN
4500 CLS:BL$="Carregar da fita"
:GOSUB 220:IF TL=1 THEN 4540
4510 LOCATE 12,12:PRINT"Confirm
a (S/N) ";
```

```
4520 R$=INKEY$:IF R$="" THEN 45
20
4530 IF R$<>"S" THEN RETURN ELS
E TL=1:GOTO 4500
4540 LOCATE 4,4:LINEINPUT"Nome
do arquivo? ";F$
4550 IF LEFT$(F$,1)<"A" OR LEFT
$(F$,1)>"Z" THEN 4540
4560 PRINT:PRINT:PRINTTAB(2);"P
osicione a fita ■ tecle <RETURN
>"
4570 IF INKEY$<>CHR$(13)THEN457
0
4580 PRINT:PRINT"Deixe o gravad
or pronto para leitura e    tecl
e <RETURN>"
4590 IF INKEY$<>CHR$(13)THEN459
0
4600 OPEN F$ FOR INPUT AS #1
4610 INPUT #1,CP,TL
4620 FOR K=1 TO TL-1:LINEINPUT
#1,TX$(K):NEXT
4630 CLOSE #1:RETURN
5500 CLS:BL$="Configura impress
ora":GOSUB 220
5510 LOCATE 0,5:PRINT"Configura
ção atual:"
5520 PRINT:PRINT:PRINT"Largura
da linha: ";MW;" col"
5530 PRINT"Largura do texto: ";
TW;" col"
5540 PRINT:PRINT"Comprimento da
 pag.: ";PL;" linhas"
5550 PRINT"Comprimento do texto
: ";TH;" linhas"
5560 PRINT:PRINT:PRINT"Quer alt
erar? ";
5570 R$=INKEY$:IF R$="" THEN 55
70
5580 IF R$<>"S" THEN RETURN
5590 CLS:GOSUB 220:LOCATE 0,8
5600 INPUT"Largura da linha ";M
W
5610 MW=INT(MW):IF MW<1 THEN 56
00
5620 INPUT"Largura do texto ";T
■
5630 TW=INT(TW):IF TW<1 ■ TW>M
W THEN 5620
5640 PRINT:INPUT"Comprimento da
 pag. ";PL
5650 PL=INT(PL):IF PL<1 THEN 56
40
5660 INPUT"Comprimento do texto
 ";TH
5670 TH=INT(TH):IF TH<1 OR TH>P
L THEN 5660
5680 GP=INT((MW-TW)/2):LF=INT((
PL-TH)/2):LF$=STRING$(LF,13)
5690 GOTO 5500
```

```
2500 PM = 9: IF CP < 9 THEN PM
= CP
2510  HTAB 1: VTAB PM + 1: PRIN
T ">"; CHR$ (8);: GET TB$
2520 CP = CP + (TB$ =  CHR$ (26
)) - (TB$ =  CHR$ (17)) + 10 *
((TB$ =  CHR$ (24)) - (TB$ =  C
HR$ (23)))
2530  IF CP < 1 THEN CP = 1
2540  IF CP > TL THEN CP = TL
2550  GOSUB 2090
2560  IF TB$ =  CHR$ (27) THEN
 RETURN
2570  IF CP > 1 AND TB$ =  CHR$
 (12) THEN TL = TL - 1: FOR K =
 CP - 1 TO TL:TX$(K) = TX$(K +
1): NEXT :TX$(TL + 1) = "":CP =
 CP - 1: GOSUB 2090
2580  IF CP > 1 AND TB$ =  CHR$
 (3) THEN A$ = " " + TX$(CP - 1
) + " ": RETURN
2590  IF TB$ =  CHR$ (16) THEN
 GOSUB 5200
2600  IF TB$ =  CHR$ (15) THEN
SF = SF + 1: IF SF = 1 THEN SS
= CP: GOTO 2620
2610  IF TB$ =  CHR$ (15) THEN
■ = CP:SF = 0: GOSUB 5300
2620  GOTO 2500
3000  RETURN : REM  LINHA TEMPO
RARIA
4000  ■ :BL$ = "GRAVAR": GOS
UB 250
4010  IF TL = 1 THEN  PRINT  CH
R$ (7);: HTAB 13: VTAB 12: PRIN
T "NADA A GRAVAR!": FOR Z = 1 T
O 2000: NEXT : RETURN
4020  HTAB 5: VTAB 5: PRINT "NO
ME DO ARQUIVO? (MAX 20 CARACT)"
: INPUT "->";F$
4030  IF  ASC (F$) < 65 OR  ASC
 (F$) > 90 THEN 4000
4040 F$ =  LEFT$ (F$,20):F$ = F
■ + ".TXT"
4050  PRINT D$;"OPEN";F$;",S";S
1;",D";S2
4060  PRINT D$;"DELETE";F$
4070  PRINT D$;"OPEN";F$
4080  PRINT D$;"WRITE";F$
4090  PRINT CP: PRINT TL
4100  FOR K = 1 TO TL - 1: PRIN
```

```
T TX$(K): NEXT
4110  PRINT D$;"CLOSE": RETURN
4500  HOME :BL$ = "CARREGAR": G
OSUB 250
4510  HTAB 5: VTAB 5: PRINT "TE
CLE [CR] PARA RETORNAR AO MENU"
: HTAB 5: PRINT "NOME DO ARQUIV
O?": INPUT "->";F$
4520  IF F$ = "" THEN  RETURN
4530  IF  ASC (F$) < 65 OR  ASC
 (F$) > 90 THEN 4500
4540 F$ = F$ + ".TXT"
4550  PRINT D$;"OPEN";F$;",S";L
1;",D";L2
4560  PRINT D$;"READ";F$
4570  INPUT CP,TL
4580  FOR K = 1 TO TL - 1
4590 TX$(K) = ""
4600  GET B$: IF B$ =  CHR$ (13
5020  VTAB 8: PRINT "CONFIGURAC
AO ATUAL:"
5030  PRINT : PRINT  TAB( 5)"CA
RREGA DE ": PRINT  TAB( 10)"SLO
) THEN  NEXT : GOTO 4620
4610 TX$(K) = TX$(K) + B$: GOTO
 4600
4620  PRINT  CHR$ (1): PRINT D$
;"CLOSE": RETURN
5000  HOME :BL$ = "CONFIGURA E/
S": GOSUB 250
T ";L1: PRINT  TAB( 10)"DRIVE "
;L2
5040  PRINT : PRINT : PRINT  TA
B( 5)"GRAVA EM ": PRINT  TAB( 1
0)"SLOT ";S1: PRINT  TAB( 10)"D
RIVE ";S2
5050  PRINT : PRINT : PRINT "QU
ER ALTERAR? ";
5060  GET R$: IF R$ <  > "S" [illegible]
D R$ <  > "N" THEN 5060
5070  IF R$ = "N" THEN  RETURN
5080  HTAB 1: VTAB 8: CALL  - 9
58: PRINT "CARREGA DE "
5090  VTAB 10: CALL  - 868: INP
UT "SLOT? ";L1
5100  IF L1 < 1 OR L1 > 7 THEN
5090
5110  VTAB 11: CALL  - 868: INP
UT "DRIVE? ";L2
5120  IF L2 <  > 1 AND L2 <  >
[illegible] THEN 5110
5130  VTAB 14: PRINT "GRAVA EM"
5140  VTAB 16: CALL  - 868: INP
UT "SLOT? ";S1
5150  IF S1 < 1 OR S1 > 7 THEN
5140
5160  VTAB 17: CALL  - 868: INP
UT "DRIVE? ";S2
5170  IF S2 <  > 1 AND S2 <  >
[illegible] THEN 5160
5180  GOTO 5020
5200  RETURN : REM  LINHA TEMPO
RARIA
5300  RETURN : REM  LINHA TEMPO
RARIA
5500  HOME :BL$ = "CONFIGURA IM
PRESSORA": GOSUB 250
5510  VTAB 5: PRINT "CONFIGURAC
AO ATUAL:"
5520  PRINT : PRINT : PRINT "LA
RGURA DA LINHA: ";MW;" COL."
5530  PRINT "LARGURA DO TEXTO:
";TW;" COL."
5540  PRINT : PRINT "COMPRIMENT
O DA PAG.: ";PL;" LINHAS"
5550  PRINT "COMPRIMENTO DO TEX
TO: ";TH;" LINHAS"
5560  PRINT : PRINT : PRINT "QU
ER ALTERAR? ";
5570  GET R$: IF R$ <  > "S" [illegible]
D R$ <  > "N" THEN 5570
5580  IF R$ = "N" THEN  RETURN
5590  HTAB 1: VTAB 5: CALL  - [illegible]
58
5600  VTAB 8: INPUT "LARGURA [illegible]
 LINHA? ";MW
5610 MW =  INT (MW): IF MW < 1
THEN 5600
5620  VTAB 9: INPUT "LARGURA DO
 TEXTO? ";TW
5630 TW =  INT (TW): IF TW < 1
OR TW > [illegible] THEN 5620
5640  VTAB 11: INPUT "COMPRIMEN
TO DA PAG.? ";PL
5650 PL =  INT (PL): IF PL < 1
THEN 5640
5660  VTAB 12: INPUT "COMPRIMEN
TO DO TEXTO? ";TH
5670 TH =  INT (TH): IF TH < 1
OR TH > PL THEN 5660
5680 GP =  INT ((MW - TW) / 2):
LF$ = "":LF =  INT ((PL - TH) /
 2)
5690  IF LF = 0 THEN 5500
5700  FOR Z = 1 TO LF:LF$ = LF$
 +  CHR$ (13): NEXT
5710  GOTO 5500
10030  PRINT  CHR$ (4);"PR#1"
10040  PRINT  CHR$ (9);"31N"
10060  LIST 5200 -
10070  PRINT  CHR$ (4);"PR#0"
```

UM SIMULADOR DE VÔO (1)

Nosso programa de simulação ■ muito semelhante aos utilizados para treinar pilotos ■ vôo por instrumentos. ■ seção apresentada neste artigo desenha ■ cabine ■ comando.

Em matéria de jogos para computador, variedade é o que não falta: você pode escolher desde a pura fantasia, indo se aventurar num mundo imaginário, até ■ simulação de situações da vida real. Este último tipo de programa permite que testemos nossa habilidade em enfrentar toda sorte de perigo sem que com isto corramos risco de vida ou de perda material.

Programas de simulação de vôo, especificamente, têm uma aplicação prática muito importante: companhias aéreas ■ escolas de aviação fazem ■ deles no treinamento de seus pilotos. Porém, não falta a tais programas um toque de fantasia: sozinho na cabine de comando, toda ■ tripulação acometida por uma doença misteriosa, você aterrissa ■ avião em segurança, usando apenas ■ das mãos.

PROGRAMAS DE TREINAMENTO

Para treinar pilotos de verdade, empregam-se simuladores totais ou "fase 3", no jargão de aviação. Estes simuladores, muito sofisticados, permitem que ■ usuário experimente todas as sensações de um piloto em situação real de vôo. Ele poderá ver ■ que o piloto vê de sua cabine, terá a impressão de aterrissar, decolar e enfrentar turbulência ■ ouvirá os sons de um vôo autêntico, inclusive ■ comandos do controle de tráfego aéreo. Teoricamente, um piloto pode completar seu treinamento em um simulador desse tipo, sem jamais ter saído do chão.

SIMULADORES DE MESA

Os programas de simulação de vôo em microcomputadores são, ■ exemplo do nosso, bem menos complexos, mas mostram-se também muito úteis para o desenvolvimento dos reflexos do piloto.

Os simuladores de mesa são essenciais para o ensino de vôo por instrumentos, um artifício que permite ao piloto dirigir o aparelho baseado apenas nos instrumentos do painel — ■ que pode ser necessário no ■ de más condições de tempo.

O QUE FAZ NOSSO SIMULADOR

Nosso programa será apresentado em três partes. Ele supõe que o piloto assumiu o comando a 2.000 metros de altitude e a 20.000 metros do centro da pista. Pela janela, pode avistar o horizonte (quando há visibilidade) e um ponto distante, que é a cabeceira da pista. Do mesmo modo que um piloto experiente, você deverá usar o raciocínio para tomar decisões corretas — baseando-se nos instrumentos do painel — ■ trazer seus passageiros ■ solo com segurança.

OS INSTRUMENTOS

O painel tem quatro mostradores. O primeiro informa ■ velocidade do vento (*airspeed*). Esta varia conforme estejamos mergulhando (a velocidade aumenta), subindo (diminui) ou mudando a potência do motor. Um contador logo abaixo do de velocidade indica ■ inclinação do vento (*bearing*).

Um segundo mostrador revela o nível do horizonte em relação ■ aeroplano. Por meio dele, mesmo sem visibilidade, saberemos onde o horizonte está. O contador logo abaixo aponta a direção da pista de pouso (*runway*).

O terceiro mostrador é um altímetro com dois ponteiros: um para centenas e outro para milhares de metros. O contador abaixo calcula ■ desvio do aparelho do centro da pista (*drift*). Como ela tem 100 metros de largura, um desvio de +50 ou -50 ao aterrissarmos será fatal. O último mostrador indica as rotações do motor por minuto (rpm). O contador abaixo fornece a distância do centro da pista.

ATERRISSAGEM

No TRS-Color, ■ imagem que vemos da cabine torna-se mais clara à medida que nos aproximamos do solo. Ao contrário dos outros micros, onde o piloto precisa se orientar pelos dados do painel, no TRS-Color pode-se observar ■ pista. Quando assumimos o controle, a pista está ao norte ■ as condições de tempo são boas. Aterrissar nestas condições é fácil, e o jogo perderia toda ■ graça se não houvesse outras situações. Para criar dificuldade, podemos mudar ■ velocidade do vento: uma rajada lateral, por exemplo, dificulta muito a aterrissagem.

O CONTROLE DO AVIÃO

O nível do controle que temos ■ simulador ■ o mesmo de que dispõe um piloto em condições reais — embora seja necessário pressionar botões ■ vez de usar um manche.

- O QUE É SIMULAÇÃO DE VÔO
- PROGRAMAS DE TREINAMENTO
- APRENDENDO A ATERRISSAR
- VÔO POR INSTRUMENTOS
- O PAINEL DO AVIÃO
- COMO CRIAR A SENSAÇÃO DE MOVIMENTO
- UMA ESCOLA DE PILOTAGEM EM CASA

Em um avião, o manche é puxado ou empurrado, movendo os estabilizadores da cauda para cima ou para baixo, conforme estejamos querendo subir ou descer. Usaremos duas teclas para obter tal efeito. A seção do programa que cuida disso será apresentada no terceiro artigo da série.

Para virar o aeroplano, o manche deve ser movido lateralmente, o que aciona os ailerons da asa. Também usaremos teclas para virar o avião.

Dois outros controles nos permitirão acelerar ou retardar o motor, o que é essencial para realizar a aterrissagem de maneira correta ou evitar que o aparelho mergulhe.

VELOCIDADE MÍNIMA

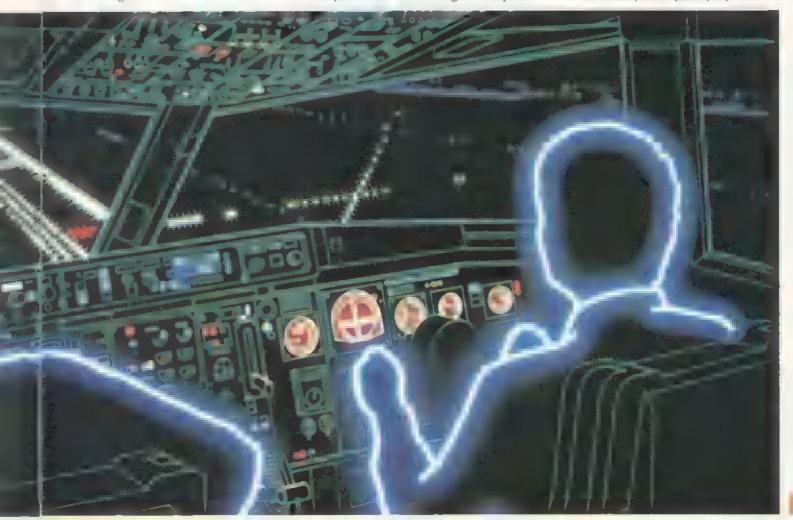
Os aviões não conseguem voar abaixo de uma certa velocidade. Em nosso programa, se a velocidade cair abaixo de 30 metros por segundo, o avião mergulhará, entrando em parafuso. Se estivermos a uma certa altitude, haverá tempo suficiente para evitar um desastre fatal, mas o perigo é iminente.

A DIVISÃO DO PROGRAMA

Dividimos o programa em três partes por ele ser muito longo e complexo.

Na primeira parte, preparamos a tela, desenhando o interior da cabine de comando, com o pára-brisa e os quatro mostradores e contadores. As legendas dos contadores e mostradores estão em inglês, como nos aviões reais.

Os comandos BASIC da listagem já são nossos conhecidos. Os usuários do TRS-Color, contudo, encontrarão um novo comando: **PCOPY**. Este é específico desta linha de micros e, mais tarde, será explicado em detalhes.

Na segunda parte do programa, tomamos os instrumentos do painel sensíveis aos movimentos do avião. Uma seção temporária fará com que o aparelho voe ao acaso, sem piloto, para que possamos

ver os instrumentos funcionando.

A parte final permitirá que você assuma o comando e teste suas habilidades na aterrissagem do avião.

A CABINE DE COMANDO

Para desenhar os instrumentos, digite a primeira parte do programa.

```
  1 POKE 23658,8
 110 GOTO 5000
5000 LET PP=-1: LET RR=-1
5010 LET C=PI/180: LET PY=-2000
: LET PZ=2000: LET AS=150
5110 PLOT 10,175: DRAW 235,0: D
RAW 0,-90: DRAW -235,0: DRAW 0,
90
5120 FOR K=0 TO 3: CIRCLE 35+K*
60,50,20: NEXT K
5130 PRINT AT 12,2;"SPEED   HOR
IZ   ALT    RPM"
5150 PRINT AT 20,0;"BEARING  RU
NWAY  DRIFT  DISTANCE"
5170 PLOT 87,50: DRAW 5,0: DRAW
 3,-3: DRAW 3,3: DRAW 5,0
5180 LET X=35: LET Y=50: GOSUB
7000: LET X=155: GOSUB 7000: LE
T X=215: GOSUB 7000
6900 STOP
7000 FOR K=0 TO 2*PI STEP PI/5:
 PLOT X+17*SIN K,Y+17*COS K: DR
AW 2*SIN K,2*COS K: NEXT K: RET
URN
```

O **POKE** da linha 1 trava o computador em letras maiúsculas. As linhas 5000 e 5010 posicionam o avião no céu: a 2.000 metros de altitude e a 20.000 metros da pista, parado no ar. As linhas que movimentam o aparelho serão apresentadas no próximo artigo.

A linha 5110 desenha a janela do avião, e a 5120, usando um laço **FOR...NEXT**, os mostradores abaixo dela. As linhas 5130 e 5150 imprimem os rótulos para os mostradores. A linha 5170 desenha um diagrama de avião no mostrador de horizonte — o horizonte artificial não será traçado por enquanto. A linha 5180 e a sub-rotina 7000 calculam as posições dos números dos mostradores: velocidade do vento, altitude e rpm. As funções **SIN** e **COS** são usadas conforme explicações dadas no artigo da página 334.

Quando você executar o programa, o interior da cabine aparecerá na tela.

```
10 SCREEN 2,2:COLOR 15,4,2
20 FOR I=0 TO 7*32*8
30 VPOKE BASE(11)+13*32*8+I,8
40 NEXT
110 GOTO 5000
4000 OPEN "GRP:" FOR OUTPUT AS
#1:PRESET(X,Y):PRINT #1,A$
4010 CLOSE #1:RETURN
5000 PP=-1:RR=-1
5010 PI=4*ATN(1):C=PI/180:PY=-2
0000:PZ=2000:VV=150
5110 LINE(10,0)-(245,80),2,B
5120 FOR K=0 TO 3:CIRCLE(35+K*6
0,128),25,2:NEXT
5130 PAINT(255,191),2
5140 X=35:Y=128:GOSUB 7000:X=15
5:GOSUB 7000:X=215:GOSUB 7000
5150 COLOR 1:X=6:Y=88:A$="AIRSP
EED":GOSUB 4000:X=78:A$="HORIZ"
:GOSUB 4000
5160 X=126:A$="ALTITUDE":GOSUB
4000:X=204:A$="RPM":GOSUB 4000
5170 COLOR 15:X=9:Y=160:A$="BEA
RING":GOSUB 4000:X=74:A$="RUNWA
Y":GOSUB 4000
5180 X=138:A$="DRIFT":GOSUB 400
0:X=188:A$="DISTANCE":GOSUB 400
0
5190 DRAW "BM81,126C10R9F5E5R9"
5500 GOTO 5500
7000 FOR K=0 TO 9:LINE(X+21*SIN
(K*PI/5),Y-21*COS(K*PI/5))-(X+1
9*SIN(K*PI/5),Y-19*COS(K*PI/5))
,2:NEXT:RETURN
```

A linha 10 seleciona a tela e suas cores. O **FOR...NEXT** das três linhas seguintes muda a cor de fundo de parte da tela para vermelho médio — **COLOR 8**. Assim, o fundo colorido dos mostradores não sofrerá a interferência de comandos gráficos do BASIC que, em geral, só atuam na cor de frente.

A linha 110 transfere o programa para a linha 5000, que estabelece os valores iniciais de algumas variáveis. Juntamente com a linha 5010, ela posiciona o avião no céu: a 20.000 metros do centro da pista e a 2.000 metros de altitude, parado no ar. As linhas que movimentam o aparelho serão apresentadas no próximo artigo.

A linha 5110 desenha a janela frontal do avião, e a linha 5120, os círculos dos mostradores. A 5130 colore todo o interior da cabine. A cor usada nessas três linhas deve ser a mesma. As linhas 5140 e 5180 rotulam os mostradores e contadores do painel. Para escrever na tela gráfica é utilizada a sub-rotina da linha 4000. Esta abre um "arquivo" com saída para a tela gráfica — **OPEN "GRP:"** — e imprime o conteúdo da variável alfanumérica A$, a partir das coordenadas X,Y.

A linha 5190 desenha um aeroplano esquemático no mostrador de horizonte. As marcas nos mostradores são desenhadas pela sub-rotina 7000.

Executando o programa agora, o interior da cabine aparecerá na tela.

```
10  HGR2 :E = 35840:T = 16384
20  FOR I = E + 32 * 8 TO E + 3
2 * 8 + 64 * 8 - 1: READ B: POK
E I,B: NEXT
100  DATA  0,0,0,0,0,0,0,0
110  DATA  8,8,8,8,0,0,8,0
120  DATA  18,18,18,0,0,0,0,0
130  DATA  18,18,63,18,63,18,18
,0
140  DATA  8 ,60 ,10 ,28 ,40 ,3
0 ,8 ,0
150  DATA  38,38,16,8,4,50,50,0
160  DATA  12,18,18,12,82,34,92
,0
170  DATA  24,16,8,0,0,0,0,0
180  DATA  32,16,8,8,8,16,32,0
190  DATA  2,4,8,8,8,4,2,0
200  DATA  0,8,42,28,28,42,8,0
210  DATA  0,8,8,62,8,8,0,0
220  DATA   0,0,0,0,0,24,16,8
230  DATA  0,0,0,62,0,0,0,0
240  DATA  0,0,0,0,0,24,24,0
250  DATA  64,32,16,8,4,2,1,0
260  DATA  28,34,38,42,50,34,28
,0
270  DATA  16,24,20,16,16,16,56
,0
280  DATA   28,34,32,24,4,2,62,
0
290  DATA  28,34,32,24,32,34,28
,0
300  DATA  16,24,20,18,62,16,16
,0
310  DATA  62,2,2,30,32,34,28,0
```

```
320  DATA  28,34,2,30,34,34,28,
0
330  DATA   62,32,32,16,8,8,8,0
340  DATA  28,34,34,28,34,34,28
,0
350  DATA  28,34,34,60,32,34,28
,0
360  DATA   0,24,24,0,0,24,24,0
370  DATA  0,24,24,0,0,24,16,8
380  DATA  32,16,8,4,8,16,32,0
390  DATA  0,0,0,62,0,62,0,0
400  DATA  4,8,16,32,16,8,4,0
410  DATA  28,34,32,16,8,8,0,8
420  DATA  28,34,58,42,58,2,60,
0
430  DATA  8,20,34,34,62,34,34,
0
440  DATA  30,34,34,30,34,34,30
,0
450  DATA  28,34,2,2,2,34,28,0
460  DATA  30,34,34,34,34,34,30
,0
470  DATA  62,2,2,62,2,2,62,0
[illegible]  DATA  62,2,2,30,2,2,2,0
490  DATA  28,34,2,58,34,34,28,
[illegible]
500  DATA  34,34,34,62,34,34,34
,0
510  DATA  24,8,8,8,8,8,28,0
520  DATA  56,16,16,16,18,18,28
,0
530  DATA  34,18,10,6,10,18,34,
```

```
0
540  DATA  2,2,2,2,2,2,62,0
550  DATA  65,99,85,73,65,65,65
,0
560  DATA  34,38,42,42,50,34,34
,0
570  DATA  28,34,34,34,34,34,28
,0
580  DATA   30,34,34,30,2,2,2,0
590  DATA  28,34,34,34,42,50,60
,64
600  DATA  30,34,34,30,18,34,34
,0
610  DATA  60,2,2,28,32,32,30,0
620  DATA  62,8,8,8,8,8,8,0
630  DATA  34,34,34,34,34,34,28
,0
640  DATA  34,34,34,34,34,20,8,
0
650  DATA  65,65,65,73,85,99,65
,0
660  DATA  34,34,20,8,20,34,34,
0
670  DATA   34,34,34,20,8,8,8,0
680  DATA  62,32,16,8,4,2,62,0
690  DATA  60,4,4,4,4,4,60,0
700  DATA  1,2,4,8,16,32,64,0
710  DATA  30,16,16,16,16,16,30
,0
720  DATA  8,20,34,0,0,0,0,0
730  DATA  0,0,0,0,0,0,62,0
740  GOTO 5000
```

```
4000 N = L * 40 + C
4010  FOR J = 1 TO  [illegible] (A$)
4020 B$ =  MID$ (A$,J,1)
4030 B =  ASC (B$)
4040 L =  INT (N / 40):C = N -
40 [illegible] L
4050  FOR I = [illegible] TO 7
4060  POKE T + (L - 8 [illegible] (L > 7)
 - [illegible] * (L > 15)) * 128 + 40 * (
L > 7) + 40 * (L > 15) + C + 10
24 * I, PEEK (E + B [illegible] [illegible] + I)
4070  NEXT I:N = N + 1: NEXT J
4080  RETURN
5000 PP =  - 1:RR =  - 1
5010 PI = [illegible] [illegible]  [illegible] (1):C = PI /
 180:C1 = PI / 10:PY =  - 20000
:PZ = 2000:AS = 150
5110  HGR2 : HCOLOR= 6: HPLOT 1
0,0 TO 276,0 TO 276,80 TO 10,80
 TO 10,0
5120  HCOLOR= 3: FOR K = 0 TO 3
: HPLOT 71 + K * 68,134: FOR I
= 0 TO 10 STEP PI / 10: HPLOT
TO 42 + K * [illegible] + 30 [illegible]  COS (C1
+ I * PI / 5),134 + 26 *  SIN (
C1 + I [illegible] PI / 5): NEXT I,K
5130 [illegible] = 2:L = 12:A$ = "AIRSPEE
D": GOSUB 4000:C = 12:A$ = "HOR
IZON": GOSUB 4000
5140 C = 22:A$ = "ALTITUDE": GO
SUB 4000:C = 33:A$ = "RPM": GOS
UB 4000
```

```
5150 C = 2:L = 21:A$ = "BEARING
": GOSUB 4000:C = 13:A$ = "RUNW
AY": GOSUB 4000
5160 C = 23:A$ = "DRIFT": GOSUB
 4000:C = 31:A$ = "DISTANCE": G
OSUB 4000
5170  HCOLOR= 5: HPLOT 85,134 T
O 104,134 TO 109,144 TO 114,134
 TO 134,134
5180  HCOLOR= 3: FOR K = ■ TO 3
: IF K = 1 THEN  NEXT ■
5190  FOR I = ■ TO 9: HPLOT 42
+ K * 68 + 28 *  COS (C1 + I *
PI / 5),134 + 24 *  SIN (C1 + I
 * PI / 5) TO 42 + K * ■■ + 26
*  COS (C1 + I * PI / 5),134 +
22 *  SIN (C1 + I * PI / 5): NE
XT I,K
```

Os usuários do TK-2000, além de não copiarem as linhas de 10 a 4080, devem fazer as seguintes modificações no programa.

```
740  GOTO 5000
4000  HTAB C + 1: VTAB L + 1: P
RINT A$
4010  RETURN
5100 MP
5120  HCOLOR= 3: FOR K = 0 TO 3
:XX = 71 + K * 68:YY = 134: FOR
 I = 0 TO 10 STEP PI / 10:X = 4
■ + K * 68 + 30 *  COS (C1 + I
* PI / 5):Y = 134 + 26 *  SIN (
C1 + I * PI / 5): HPLOT XX,YY T
O X,Y:XX = X:YY = Y: NEXT I,K
```

As linhas iniciais criam um banco de blocos no topo da memória, para que o Apple possa escrever na tela. Essas linhas já foram apresentadas em artigo anterior, quando explicamos o uso de blocos gráficos: pode ser, portanto, que você não precise digitá-las novamente. O TK-2000 pode escrever na tela gráfica sem esse artifício; assim, seus usuários devem começar a digitação a partir da linha 5000.

Como vamos utilizar a página 2, armazenamos o banco de blocos em outra área da memória, devidamente reservada por meio de **HIMEM:**.

A sub-rotina que começa na linha 4000, responsável pela impressão de palavras na tela gráfica, é essencialmente a mesma publicada em artigo anterior, só que com outro número de linhas. Os usuários do TK-2000 dispõem de uma versão bem mais simples dessa sub-rotina. Na realidade, ela poderia ser dispensada neste micro; resolvemos introduzi-la aqui para que os programas do Apple e do TK-2000 não ficassem muito diferentes.

As linhas 5000 e 5010 posicionam o avião no céu: a 20.000 metros do centro da pista e a 2.000 metros de altitude, parado no ar. As linhas que movimentam o aparelho serão apresentadas no próximo artigo.

A linha 5110 desenha o pára-brisa do avião. No TK-2000 há um comando **MP** para ativar a segunda página de vídeo; no Apple, isso é feito por **HGR2**. Note que o programa do TK-2000 deve conter **HGR2** também, para permitir o uso das linhas inferiores da tela.

A linha 5120, um pouco diferente para os dois micros, desenha os círculos dos mostradores. As linhas 5130 a 5160 rotulam os mostradores e contadores do painel em inglês, como nos aviões reais. A 5170 traça o diagrama de um aeroplano no mostrador de horizonte. As marcas nos mostradores são feitas pelas linhas 5180 e 5190.

Ao ser executado, o programa desenhará o interior da cabine de nosso aparelho imaginário.

```
10 PCLEAR 8:PMODE 4,1
20 DIM LE$(26)
30 FOR K=0 TO 26:READ LE$(K):NE
XT
40 FOR K=0 TO 9:READ NU$(K):NEX
T
50 DATA BR2,ND4R3D2NL3ND2BE2,ND
4R3DGNL2FDNL3BU4BR2,NR3D4R3BU4B
R2,ND4R2FD2GL2BE4BR,NR3D2NR2D2R
3BU4BR2
60 DATA NR3D2NR2D2BE4BR,NR3D4R3
U2LBE2BR,D4BR3U2NL3U2BR2,ND4BR2
,BD4REU3L2R3BR2,D2ND2NF2E2BR2
70 DATA D4R3BU4BR2,ND4FREND4BR2
,ND4F3DU4BR2,NR3D4R3U4BR2,ND4R3
D2NL3BE2,NR3D4R3NHU4BR2
80 DATA ND4R3D2L2F2BU4BR2,BD4R3
U2L3U2R3BR2,RND4RBR2,D4R2U4BR2,
D3FEU3BR2,D4EFU4BR2
90 DATA DF2DBL2UE2UBR2,DFND2EUB
R2,R3G3DR3BU4BR2
100 DATA NR2D4R2U4BR2,BDEND4BR2
,R2D2L2D2R2BU4BR2,NR2BD2NR2BD2R
2U4BR2,D2R2D2U4BR2,NR2D2R2D2L2B
E4,D4R2U2L2BE2BR2,R2ND4BR2,NR2D
4R2U2NL2U2BR2,NR2D2R2D2U4BR2
110 GOTO 5000
4000 FOR K=1 TO LEN(A$)
4010 B$=MID$(A$,K,1)
4020 IF B$>="0" AND B$<="9" THE
N DRAW NU$(VAL(B$)):GOTO 4050
4030 IF B$=" " THEN N=0 ELSE N=
ASC(B$)-64
4040 DRAW LE$(N)
4050 NEXT :RETURN
5000 PP=-1:RR=-1
5010 PI=4*ATN(1):C=PI/180:PY=-2
0000:PZ=2000:VV=150
5110 PCLS:LINE(10,0)-(245,80),P
SET,B
5120 FOR K=0 TO 3:CIRCLE(35+K*6
0,120),25,5:NEXT
5130 DRAW "BM18,88S4":A$="AIRSP
EED":GOSUB 4000:DRAW "BM80,88":
A$="HORIZON":GOSUB 4000
5140 DRAW "BM140,88":A$="ALTITU
DE":GOSUB 4000:DRAW"BM208,88":A
$="RPM":GOSUB 4000
5150 DRAW "BM18,160":A$="BEARIN
G":GOSUB 4000:DRAW"BM82,152":A$
="RUNWAY":GOSUB 4000:DRAW"BM80,
160":A$="BEARING":GOSUB 4000
5160 DRAW"BM144,160":A$="DRIFT"
:GOSUB 4000:DRAW"BM200,160":A$=
"DISTANCE":GOSUB 4000
5170 DRAW"BM81,118R9F5E5R9"
5180 X=35:Y=120:GOSUB 7000:X=15
5:GOSUB 7000:X=215:GOSUB 7000
5190 PCOPY 3 TO 5:PCOPY 3 TO 7:
PCOPY 4 TO 6:PCOPY ■ TO 8:SCREE
N 1,1
5500 GOTO 5500
7000 FOR K=0 TO 9:LINE(X+24*SIN
(K*PI/5),Y-24*COS(K*PI/5))-(X+2
1*SIN(K*PI/5),Y-21*COS(K*PI/5))
,PSET:NEXT:RETURN
```

A primeira parte do simulador inclui um comando ainda não utilizado em INPUT: **PCOPY**, que assegura a movimentação suave da imagem na tela.

As linhas 20 a 110 preparam as matrizes que não contêm os blocos gráficos necessários ao desenho da cabine. A sub-rotina das linhas 4000 e 4050 imprime os gráficos na tela.

As linhas 5000 e 5010 posicionam o aparelho no céu: a 20.000 metros do centro da pista e a 2.000 metros de altitude, parado no ar. As linhas que movimentam o aparelho serão apresentadas no próximo artigo.

A linha 5110 desenha o pára-brisa, 5120 desenha os círculos dos mostradores, e as linhas 5130 a 5160 rotulam os mesmos — em inglês, como nos aviões reais. A linha 5170 coloca o diagrama de um avião no mostrador de horizonte. As marcas nos mostradores são desenhadas pela linha 5180 e pela sub-rotina da linha 7000.

Os comandos **PCOPY** na linha 5190 desenham gráficos em páginas ainda invisíveis e depois copiam-nos na tela. Assim, auxiliam a preservar os desenhos do fundo, que são atualizados antes de serem transferidos para a tela, de modo que o lapso de tempo entre um quadro e outro seja mínimo. Isso significa que, quando o aeroplano está em movimento, os mostradores são alterados de acordo, simultaneamente. Sem **PCOPY**, apenas um mostrador poderia ser atualizado de cada vez.

Ao executar esta seção do programa, o interior da cabine aparecerá desenhado na tela.

No próximo artigo, apresentaremos as linhas que fazem o avião voar, embora ainda sem controle. Ele vagará pelos céus, mergulhando e subindo ao acaso. Você poderá controlá-lo quando concluir o programa, no terceiro e último artigo. Finalmente, a vida dos passageiros estará em suas mãos.

AMPLIE O BASIC DO TRS-COLOR

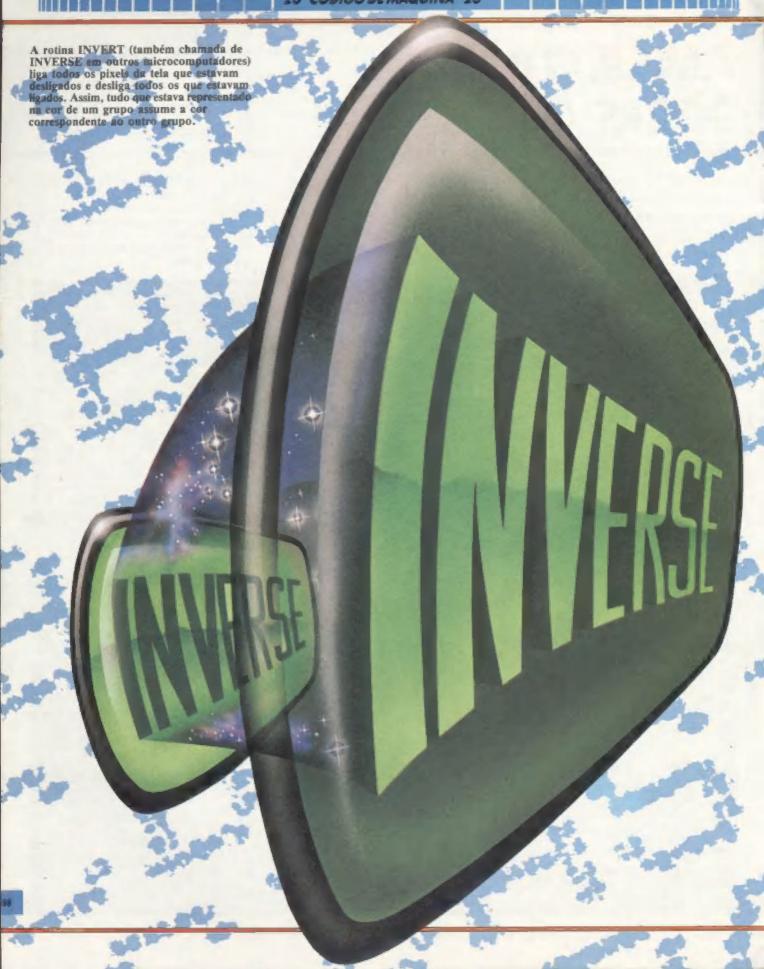
- PLANEJAMENTO DAS NOVAS INSTRUÇÕES
- O QUE É UM **STUB**
- NOVOS COMANDOS NO MICRO
- RECUPERAÇÃO DE PROGRAMAS

O BASIC do TRS-Color não é intocável. Pode-se modificá-lo para incluir novos comandos, o que é útil sobretudo quando o micro executa tarefas que empregam certas rotinas repetidas vezes.

O BASIC é apenas um programa em linguagem de máquina funcionando em seu microcomputador. Ainda que as instruções executadas estejam codificadas na memória ROM, no TRS-Color é possível fazer algumas modificações para criar novos comandos.

Esta possibilidade revela-se bastante útil nos casos em que usamos o computador para realizar uma tarefa específica que emprega determinadas rotinas repetidas vezes. Neste artigo, criaremos duas novas instruções destinadas ao micro TRS-Color.

A rotina INVERT (também chamada de INVERSE em outros microcomputadores) liga todos os pixels da tela que estavam desligados e desliga todos os que estavam ligados. Assim, tudo que estava representado na cor de um grupo assume a cor correspondente ao outro grupo.

T

A rotina que se segue acrescenta dois comandos ao BASIC do TRS-Color: **OLD**, que permite a recuperação de um programa apagado acidentalmente com **NEW**, e **INVERT**, que troca o grupo de quatro cores que estiver sendo utilizado na tela gráfica.

```
10   ORG 31000
20   SETUP LDX #298
30   LDU #308
40   STONE LDA ,X+
50   STA ,U+
60   CMPX #308
70   BLO STONE
80   LDA #2
90   STA 298
100   LDX #NEWRDS
110   STX 299
120   LDX #NEWDSP
130   STX 301
140   LDX #NEWUSR
150   STX 176
160   LDU #$8B8D
170   LDA #10
180   STTWO STU ,X++
190   DECA
200   BNE STTWO
210   RTS
220   NEWRDS FCB 79,76,196
230   FCB 73,78,86,69,82,212
240   NEWDSP CMPA #$CE
250   BLO NDONE
260   CMPA #$D0
270   BHS NDONE
280   SUBA #$CE
290   LDX #NEWTBL
300   JMP $84ED
310   NDONE JMP $89B4
320   OLD LDU 25
330   PSHS U
340   LEAX 4,U
350   OLDONE LDA ,X+
360   BNE OLDONE
370   TFR X,D
380   SUBD ,S++
390   TSTA
400   BEQ OLDTWO
410   JMP $8B8D
420   OLDTWO STX ,U
430   OLDTHR TFR X,U
440   LDX ,U
450   BNE OLDTHR
460   LEAU 2,U
470   STU 27
480   STU 29
490   STU 31
500   RTS
510   INVERT LDA 65314
520   EORA #8
530   STA 65314
540   RTS
550   NEWTBL FDB $7964
560   FDB $7989
570   NEWUSR EQU *
```

O QUE É UM STUB

Para acrescentar comandos ao programa BASIC, é necessário que reservemos as palavras correspondentes. Temos, assim, que ampliar a lista de palavras reservadas, bem como dirigir o BASIC aos programas em código que executam as novas funções.

A posição das rotinas que executam os comandos BASIC é apontada por determinados vetores — ou apontadores — que ficam em porções da memória denominadas *stubs*. Existem normalmente dois stubs, cada um ocupando dez bytes da memória RAM. O segundo deles, porém, é só um sinalizador de final de tabela, não tendo outra função.

Quando acrescentamos instruções devemos criar também um stub que aponte para as novas rotinas. Mas, antes disso, é preciso deslocar o stub que indica o fim da tabela para uma posição mais alta na memória, criando o espaço necessário.

O endereço inicial do segundo stub é 298, e as primeiras seis instruções do programa aqui apresentado transferem-no para 308. O registro **X** é usado como apontador para o stub original, enquanto o registro **U** aponta para sua nova posição. Os operandos **,X +** e **,U +** incrementam esses apontadores a cada volta do laço **STONE**.

Este programa não pode ser usado em sistemas com disquete, pois os comandos de disco usam o mesmo stub.

CRIAÇÃO DE UM NOVO STUB

Uma vez garantido o espaço necessário na memória de acesso aleatório e terminado o laço **STONE**, passamos à criação do novo stub.

O primeiro byte desse novo stub corresponde ao número de instruções apontadas por ele. Portanto, **LDA #2** e **STA 298** colocam o número 2 no primeiro byte.

Os dois próximos bytes trazem o endereço da tabela de nomes de comandos. Estes nomes aparecem imediatamente após o rótulo **NEWRDS**, com suas letras codificadas uma a uma em ASCII. Só não ocorre o mesmo com a última letra da cada palavra, que se distingue por ter seu bit mais significativo ativado. Assim, D — última letra do comando **OLD** — é representada pelo seu código ASCII, 68, mais 10000000 em binário ou 128 em decimal (68 + 128 = 196). Da mesma maneira, o T de **INVERT** será 212 e não 84.

Finalmente, quarto o e o quinto bytes trazem o endereço inicial das rotinas que executam os comandos. Esse endereço é encontrado nos bytes após o rótulo **NEWTBL**.

ADICIONE SEUS PRÓPRIOS COMANDOS

Depois de entender como funcionam os *stubs* no reconhecimento dos comandos BASIC, o programador pode criar seus próprios comandos. Quem já tem uma certa experiência deve começar pelos comandos sem operandos, como os deste artigo. Qualquer rotina pode se transformar num comando desse tipo. Basta o BASIC encontrar o novo comando em uma linha de programa, e o controle é desviado para a rotina em código cujo endereço está no *stub*. Após sua execução, o controle retorna ao interpretador.

Comandos que operam sobre números ou variáveis exigem muito do programador. Se os números não forem inteiros, ele terá de conhecer os segredos da aritmética de ponto flutuante e dos números decimais codificados em binário. Além de um bom conhecimento de linguagem de máquina, é preciso um profundo domínio da arquitetura do TRS-Color; ao mesmo tempo, deve-se saber onde ficam as variáveis do sistema, como são armazenadas as variáveis e as linhas BASIC, e onde podem ser colocados temporariamente os valores dos operandos.

VETORES USR

O vetor que fica nos endereços 176 e 177 aponta para as posições que contêm o endereço de rotinas **USR**. Estas posições ficam geralmente logo acima do segundo stub, em uma tabela que começa em 308. O segundo stub, contudo, foi empurrado para dentro dessa área para criar espaço para o novo stub, de forma que a área **USR** também precisa ser relocada.

LDX #NEWUSR e **STX 176** colocam o endereço do rótulo **NEWUSR** nas posições 176 e 177. **EQU ***, que fica logo após **NEWUSR**, no final do programa, reserva os próximos bytes para a tabela **USR**.

OS CÓDIGOS DOS NOVOS COMANDOS

O código de maior valor usado pelo BASIC é CD. Assim, os códigos dos dois novos comandos serão CE e CF.

CMPA #$CE e **BLO NDONE** verificam se o código da instrução está abaixo do valor dos novos códigos. Se isso ocorrer e o primeiro stub não tiver reconhecido o comando, houve um erro

de sintaxe. O microprocessador é, então, enviado ao rótulo **NDONE** que, por sua vez, provoca um salto para o endereço 89B4 — uma sub-rotina da ROM que emite uma mensagem de erro.

CMPA #50 e **BHS NDONE** fazem o mesmo se o código for maior que CF, ou seja, se ele estiver acima da faixa em que ficam os códigos das novas instruções.

Se o código passar por esses dois testes, fica comprovado que ele corresponde a um dos novos comandos. O programa, então, segue em frente. CE é subtraído do valor do código, que está em A. O resultado permanece em A.

O endereço inicial da tabela de endereços dos novos comandos é colocado em X. Assim, quando o processador vai para 84ED — uma rotina que cuida do reconhecimento dos vários comandos BASIC —, leva consigo os valores contidos em X e em A.

A ROTINA DO COMANDO OLD

Quando usamos **NEW** para apagar um programa, os valores de diversas variáveis do sistema modificam-se. Se restaurarmos os valores dessas variáveis antes de digitarmos outro programa ou de usarmos um comando **PCLEAR**, poderemos recuperar o programa antigo. O comando **NEW** modifica os dois primeiros bytes da primeira linha do programa que se quer recuperar. Esses bytes indicam o endereço inicial da linha seguinte. Outras variáveis do sistema que apontam o final da área do programa BASIC também são modificadas. O que o comando **OLD** faz é restaurar o valor original de todos esses apontadores.

A rotina que cuida disso começa com a instrução **LDU 25**, que coloca o conteúdo das posições 25 e 26 no registro U. Elas contêm o endereço inicial do programa BASIC. **PSHS U** coloca este valor na pilha, para que possamos recuperá-lo mais tarde.

A instrução **LEAX 4,U** cuida do endereço do byte 4, contado a partir do início do BASIC. Os dois primeiros bytes — 0 e 1 — continham o endereço da próxima linha. Os dois seguintes — 2 e 3 —, o número da linha. Assim, U contém agora o primeiro byte da linha propriamente dita.

LDA ,X+ coloca este byte no acumulador e incrementa X. **BNE OLDONE** faz o processador repetir essa instrução até encontrar um zero, que sinaliza o final da linha BASIC.

Quando o programa encontra um zero e sai do laço, X já foi incrementado e aponta para o endereço inicial da próxima linha BASIC.

VERIFICAÇÃO DE ERROS

Se o comando **OLD** for acionado sem que haja um programa BASIC aproveitável na memória, devemos impedir que o lixo que porventura esteja na área BASIC seja recuperado. Precisamos, assim, de um teste que verifique se a primeira linha ainda faz sentido.

Para fazer isso, o conteúdo do apontador X é colocado em D e dele subtraímos o último item da pilha — o endereço inicial do BASIC — com **SUBD ,S++**. O resultado indica o comprimento da primeira linha, que permanece armazenada em D. O apontador da pilha é decrementado, o que retira o endereço inicial do BASIC da pilha.

TSTA testa o acumulador A, isto é, verifica o conteúdo do byte mais significativo de D (os registros A e B juntos constituem o registro D) e estabelece os sinalizadores de acordo. **BEQ OLDTWO** salta as instruções seguintes, se o sinalizador zero foi ativado. Se o conteúdo de A não for zero, o registro D continha um valor maior que 255 — o maior número de bytes que uma linha BASIC pode conter. Nesse caso, o processador é enviado para a rotina da memória ROM, em 8BBD, que emite uma mensagem de erro.

RESTAURE OS APONTADORES

O registro U aponta para o primeiro byte da primeira linha BASIC. Assim, para restaurar o apontador do início da linha seguinte, colocamos o valor de X no endereço apontado por U, usando **STX ,U**.

O passo seguinte é descobrir onde fica o final da área do programa BASIC. O laço **OLDTHR** trata disso.

Como os dois primeiros bytes de qualquer linha apontam para o início da seguinte, é fácil saltar de linha em linha, transferindo o conteúdo de X para U e colocando em X o valor contido nas posições apontadas por U. X contém o endereço da próxima linha, que é colcado em U. U aponta, então, para os dois bytes que contêm o endereço da linha que fica ainda mais adiante; este valor é colocado em X.

O mesmo laço é executado repetidas vezes, até que se encontre uma linha cujos dois primeiros bytes sejam 00 00. Este é o final do programa BASIC. Quando ele é encontrado, o processador sai do laço.

O endereço que fica logo acima desses dois últimos bytes é colocado em U, que passa a apontar para o início da área das variáveis.

STU 27, **STU 29**, e **STU 31** copiam esse endereço nas variáveis do sistema que ficam em 28 e 29, apontando para o início da área de variáveis; em 29 e 30, apontando para o início da tabela de matrizes; e em 31 e 32, apontando para o topo da RAM usada.

RTS retorna ao BASIC.

A ROTINA INVERT

A posição de memória FF22 fica na memória de entrada e saída — I/O — de periféricos e controla a saída de dados gráficos para a tela. O bit 3 dessa posição determina o grupo de cores que está sendo utilizado.

LDA $FF22 coloca o conteúdo daquele endereço no acumulador e o bit 3 é alterado pela operação lógica "ou exclusivo", entre A e 8 (00001000, em binário). O resultado é recolocado em FF22, e **RTS** volta ao BASIC.

Essa rotina simplesmente alterna dois grupos de cores, fazendo com que tudo o que está representado na cor de um grupo assuma a cor correspondente do outro grupo de cores.

O programa não é recolocável.

Só há uma forma de o micro aceitar instruções adicionais?

Não. Embora os diversos modelos de micro tenham um BASIC semelhante, o programa que o entende varia muito de um para outro: alguns não têm *stubs*, e novas instruções devem ser adicionadas sem estes.

Podemos chamar de "sistema" o programa existente na ROM que entende o BASIC. Ele deve ser muito bem organizado para poder associar a cada comando a rotina em código certa (muitas vezes os comandos agem sobre operandos, o que complica ainda mais a tarefa do sistema). Assim, ele mantém diversas tabelas com endereços e valores que o orientam na interpretação do BASIC. Os *stubs* nada mais são que tabelas desse tipo. Mas existem outras. A principal maneira de se adicionar novos comandos ao BASIC de uma máquina consiste em alterar determinados valores nessas tabelas de controle.